



Qualcomm Technologies International, Ltd.

Active Noise Cancellation for Qualcomm S3 Gen 1/2/3

User Guide

80-54357-1 Rev. AD

September 13, 2024

Confidential – Qualcomm Technologies, Inc. and/or its affiliated companies – May Contain Trade Secrets

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to DocCtrlAgent@qualcomm.com.

Qualcomm Technologies International, Ltd. is a company registered in England and Wales
with a registered office at: Churchill House, Cambridge Business Park, Cowley Road,
Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

© Qualcomm Technologies, Inc. and/or its subsidiaries. All rights reserved.

Revision history

Revision	Date	Description
AA	February 2023	Initial release.
AB	July 2023	Updates for ADK 23.1
AC	January 2024	Updated <i>Build and configure the Adverse Acoustic Handling capability</i> section. Updated <i>Storing device-specific ANC fine gain</i> section.
AD	September 2024	Updated the following sections: ▪ <i>Enhanced ANC mode</i> ▪ <i>Wind Noise Detect feature</i> ▪ <i>NoiseID feature overview</i>

Contents

Revision history	2
1 Qualcomm® Active Noise Cancellation for Earbuds Application	15
1.1 Supported chipsets	15
2 ANC architecture and features	17
2.1 ANC architecture	18
2.2 ANC hardware block	19
2.3 ANC modes	20
2.4 ANC use cases	26
2.5 Adaptive ANC capability	30
2.6 Support for Custom Adaptive ANC	30
2.6.1 Creating custom Adaptive ANC capability	36
3 ANC hardware specification	37
3.1 ANC filter parameters	38
3.2 ANC full-band monitor mux	42
4 Understanding microphone configurations	44
4.1 ANC microphone configuration guidelines	44
4.2 Stereo headset Qualcomm hybrid ANC use case (Static ANC)	45
4.3 Mono earbud Qualcomm hybrid ANC use case (Static, Adaptive ANC, and Enhanced Adaptive ANC) ...	46
5 ANC development process	47
5.1 ANC tuning setup	49
6 Configure ANC in the Earbuds Application	52
6.1 Enable ANC features in the Earbuds application	53
6.2 Configure digital microphones	54
6.3 Route hardware mic instances to software mic instances	55
6.4 Connect software mics to ANC mics	56
6.5 Assign microphones for Qualcomm ANC or enhanced ANC (Run mode)	58
6.6 Assign microphones for Qualcomm ANC or enhanced ANC (Tuning mode)	58
6.7 Configure ANC modes for the required use cases	59

6.8 Configure IIR filter sample rate	60
6.9 Configure the filesystem for ANC or enhanced ANC	61
6.10 Configure filesystem for Qualcomm Adaptive ANC or enhanced Adaptive ANC	61
6.11 Enable dynamic ANC filter transition	61
6.12 Build and deploy the application	62
7 Perform audio curation and debugging	63
7.1 Debugging with Pydbg	63
7.2 GAIA application for audio curation	65
8 Adaptive ANC and Adaptive Ambient tuning using Qualcomm Audio Calibration Tool	66
8.1 Prerequisites for QACT tuning process	67
8.2 Connect the device to QACT	67
8.3 Adaptive ANC tuning	69
8.3.1 Prerequisites for tuning Adaptive ANC	69
8.3.2 Preparing a new device for tuning	70
8.3.3 Tunable parameters	73
8.3.4 FxLMS tuning: Configure Convergence speed and Converged value	82
8.3.5 Tuning Adaptive ANC when the adaptation rate is not optimum	86
8.4 Adaptive Ambient tuning	88
9 Static ANC tuning using ANC Filter Designer	93
9.1 ANC Tuning mode vs Mission mode	93
9.2 Launch the ANC Filter Designer	94
9.3 Connect to the DUT in Tuning mode	94
9.4 Connect to the DUT in Mission mode	96
9.5 ANC Filter Designer configuration	99
9.5.1 ANC filter design modes	100
9.5.2 ANC Filter Designer mode parameters	101
9.6 Access device with Wireless Debug	102
9.7 Generate acoustic path models	103
9.7.1 Collect recordings in Tuning mode	103
9.7.2 Collect recordings in Mission mode	109
9.7.3 Tune filters for models	111
9.7.4 Generate reference models in Tuning mode	114
9.7.5 Generate reference models in Mission mode	116
9.7.6 Import, export, and remove reference models	118
9.7.7 Load a model generated in other tools	119
9.8 Modify PEQ bands	119
9.9 Feedback ANC filter design	119

9.9.1 Configure feedback ANC	120
9.9.2 Automatic feedback tuning	121
9.9.3 Advanced feedback tuning	122
9.10 Feed-forward ANC filter design	124
9.10.1 Configure feed-forward ANC	125
9.10.2 Automatic feed-forward tuning	126
9.10.3 Advanced feed-forward tuning	127
9.11 Echo canceller filter design	128
9.11.1 Configure echo canceler ANC	129
9.11.2 Automatic echo canceller tuning	130
9.12 Leak-through tuning	131
9.12.1 Automatic leak-through tuning	132
9.13 ANC Filter Designer plot view	132
9.14 Save the tuning and workspace	133
9.14.1 Snapshots	134
9.14.2 Load/save HTF file	135
9.14.3 Save workspace	137
9.14.4 Read and write from persistence	137
9.14.5 Export the IIR filter	139
9.15 Configure frontend gains	140
10 Production-line calibration	141
10.1 Tuning setup and process	142
10.2 Hardware and software requirements for deploying ANC	143
10.3 Wireless calibration	143
10.3.1 Security advisory	143
10.3.2 ANC test setup for fine tuning	144
10.3.3 ANC production calibration configuration	145
10.4 Wired calibration	155
10.4.1 Develop production-line test application	155
10.4.2 Determine calibrated gain parameters	156
10.4.3 Gain parameters and calculation	162
11 Adaptive ANC tuning use cases	169
11.1 Earbud Fit Test feature	169
11.1.1 Build and configure the Earbud Fit Test capability	169
11.1.2 Source file design for Earbuds Fit Test	173
11.1.3 Tune the wear status graph	175
11.2 Wind Noise Detect feature	183

11.2.1 Wind Noise Detect (WND) capability modes	184
11.2.2 Wind noise mitigation	184
11.2.3 Build and configure the Wind Noise Detection capability	184
11.2.4 Tune the Wind Noise Detection feature	186
11.3 Auto Transparency feature	191
11.3.1 ATR_VAD capability	192
11.3.2 Build and configure the ATR_VAD capability	192
11.3.3 Tune the Auto Transparency feature	194
11.4 Adverse Acoustic Event Handler (AAH) feature	198
11.4.1 Adverse Acoustic Handling capability system	199
11.4.2 Build and configure the Adverse Acoustic Handling capability	199
11.4.3 Tune the Adverse Acoustic Handling capability	200
11.4.4 Statistics of Adverse Acoustic Event Handler capability	202
11.5 DSP SW based leak-through feature	202
11.5.1 Pydbg commands to control DSP-based audio leak-through	203
11.5.2 Enable, disable, or toggle audio leak-through functionality	203
11.6 NoiseID feature overview	207
11.6.1 Build Earbud app with NoiseID	211
11.6.2 Pydbg commands for NoiseID	212
11.6.3 Tune NoiseID	212
A ANC stream APIs	215
Terms and definitions	220
Document references	222

Tables

Table 1-1: Supported chipset platform.....	15
Table 2-1: Programmable paths associated with ANC hardware block.....	19
Table 2-2: Adaptive ANC playpen files.....	30
Table 3-1: QCC517x DC filter cut-off frequencies for the recommended ANC sampling rates.....	38
Table 3-2: QCC517x low pass filter cut-off frequencies for the recommended ANC sampling rates.....	39
Table 3-3: Example fine gains.....	40
Table 4-1: Rules for microphone and ANC configurations.....	44
Table 4-2: Stereo headset hybrid ANC use case.....	45
Table 4-3: Mono Earbud Qualcomm hybrid ANC use case.....	46
Table 5-1: Subsystems on a QCC variant.....	50
Table 5-2: Software components running on a QCC variant.....	50
Table 6-1: Definitions for enabling Qualcomm ANC, Enhanced ANC, Adaptive ANC, and Enhanced Adaptive ANC.....	53
Table 6-2: Digital microphone parameters.....	54
Table 6-3: ANC microphone settings.....	55
Table 6-4: Settings for connecting microphones.....	56
Table 6-5: ANC microphone audio instances.....	59
Table 6-6: AHM capability parameters.....	62
Table 7-1: Debugging with Pydbg.....	63
Table 8-1: QACT file paths.....	67
Table 9-1: Pydbg commands used in Tuning mode.....	96
Table 9-2: Pydbg commands used in Mission mode.....	98
Table 9-3: ANC Filter Designer mode parameters.....	101
Table 9-4: Reference model file format.....	119

Table 9-5: Feedback ANC parameters.....	121
Table 9-6: Automatic feed-forward tuning parameters.....	126
Table 9-7: ANC Filter Designer plot legend.....	132
Table 10-1: Hardware and software requirements.....	143
Table 10-2: Coarse value to dB and Q6.9 format mapping.....	149
Table 10-3: AT commands to the device.....	149
Table 10-4: AT commands from the device.....	151
Table 10-5: TestEngine files and their description.....	155
Table 10-6: ANC API functions.....	157
Table 10-7: ANC API: Gain setting functions.....	157
Table 10-8: ANC API: Functions for reading and writing parameters.....	158
Table 10-9: Read/write scripts and their usages.....	161
Table 10-10: Python ANC API variables.....	162
Table 10-11: ANC block gain parameters.....	163
Table 11-1: Pydbg test commands for Earbuds Fit Test.....	170
Table 11-2: AANC parameters to configure touch sensitivity of Earbuds.....	178
Table 11-3: Pydbg WND test commands.....	185
Table 11-4: Tunable parameters.....	186
Table 11-5: Wind intensity parameters.....	187
Table 11-6: Wind mitigation parameters.....	189
Table 11-7: Gain payload.....	190
Table 11-8: Wind messaging parameters.....	191
Table 11-9: Pydbg ATR_VAD test commands.....	192
Table 11-10: Tunable parameters.....	196
Table 11-11: VAD detection flag tuning parameters.....	197
Table 11-12: ATR_VAD messaging tunable parameters.....	198
Table 11-13: AHM tuning parameters.....	200
Table 11-14: AAH algorithm tuning parameters.....	201
Table 11-15: AAH timing tuning parameters.....	201
Table 11-16: ANC proxy filter tunable parameters.....	202
Table 11-17: AAH statistics.....	202

Table 11-18: Pydbg commands for NosielD.....	212
Table 11-19: Tuning parameters for type-based classification.....	213
Table 11-20: Tuning parameters for level-based classification.....	213

Figures

Figure 2-1: Top-level logical diagram of the ANC architecture.....	18
Figure 2-2: ANC hardware block.....	19
Figure 2-3: Feed-forward system.....	20
Figure 2-4: Feed-forward ANC.....	21
Figure 2-5: Feedback ANC system.....	22
Figure 2-6: Feedback ANC.....	22
Figure 2-7: Enhanced ANC mode.....	23
Figure 2-8: Adaptive ANC in single mode.....	24
Figure 2-9: Adaptive ANC in Enhanced mode.....	24
Figure 2-10: Adaptive ANC in Enhanced Dual mode.....	25
Figure 2-11: Logical diagram of ANC tuning use case.....	27
Figure 2-12: KSE simulation graph for Enhanced ANC.....	33
Figure 2-13: KSE simulation graph for dual filter topology.....	34
Figure 3-1: Enhanced ANC hardware and signal routing without RxMix.....	37
Figure 3-2: IIR coefficient equations.....	39
Figure 3-3: Enhanced ANC hardware and signal routing with RxMix.....	42
Figure 5-1: ANC development process and tools.....	47
Figure 5-2: High-level ANC interface block diagram.....	49
Figure 6-1: ANC configuration process.....	52
Figure 6-2: AHM capability parameters.....	62
Figure 8-1: QACT User Interface.....	66
Figure 8-2: kalaccess.dll and workspace file path.....	68
Figure 8-3: Connect device.....	68
Figure 8-4: Enable statistics in QACT.....	70

Figure 8-5: Disable energy detectors in QACT.....	70
Figure 8-6: Adaptive gain statistic displayed in QACT.....	71
Figure 8-7: Change FxLMS Mu parameter in QACT.....	71
Figure 8-8: Change the coarse gain in static mode from QACT.....	72
Figure 8-9: Disable mode override to re-test tuning in QACT.....	72
Figure 8-10: AANC CONFIG flags displayed in QACT.....	73
Figure 8-11: AANC debug configuration flags displayed in the QACT generic view.....	75
Figure 8-12: Updating FxLMS Mu parameter in QACT.....	83
Figure 8-13: Gain convergence with default value.....	83
Figure 8-14: Gain convergence with increased FXLMS_MU value.....	84
Figure 8-15: Updating FxLMS Lambda parameter in QACT.....	85
Figure 8-16: Gain converge variation with FXLMS_MU value.....	86
Figure 8-17: Measuring rate of gain convergence.....	87
Figure 8-18: Finding sweet spot for gain convergence.....	87
Figure 8-19: ANC Compander tuning interface.....	89
Figure 8-20: HCGR gain recovery.....	92
Figure 9-1: ANC connected to the DUT in Tuning mode.....	94
Figure 9-2: General tab in ANC Configuration dialog.....	95
Figure 9-3: ANC Device Configuration in Tuning mode.....	95
Figure 9-4: ANC Filter Designer connected to the device in Tuning mode.....	96
Figure 9-5: General tab in ANC Configuration dialog.....	97
Figure 9-6: ANC Device Configuration Dialog in Mission mode.....	97
Figure 9-7: ANC Filter Designer connected to the device in Mission mode.....	98
Figure 9-8: Opening the Configuration dialog in the ANC Filter Designer.....	99
Figure 9-9: General configuration dialog.....	100
Figure 9-10: Wireless Debug Test Commands dialog.....	103
Figure 9-11: Tuning mode chamber setup.....	104
Figure 9-12: Selecting S-path in Tuning mode Path Recording dialog.....	105
Figure 9-13: S-path recording.....	105
Figure 9-14: Selecting P-path Ext in Tuning Mode Path Recording dialog.....	106
Figure 9-15: P-path recording.....	107

Figure 9-16: P-path recording in Tuning mode example.....	108
Figure 9-17: S-path recording in Tuning mode example.....	108
Figure 9-18: E-path recording in Tuning mode example.....	109
Figure 9-19: Mission mode chamber setup.....	110
Figure 9-20: Example recording in Mission mode.....	111
Figure 9-21: Pre-tune feed-forward recording in Mission mode.....	112
Figure 9-22: Feed-forward performance in Mission mode.....	112
Figure 9-23: Pre-tune feedback recording in Mission mode.....	113
Figure 9-24: Feedback performance in Mission mode.....	114
Figure 9-25: Generate a reference model in Tuning mode menu.....	114
Figure 9-26: Generate a reference model in Tuning mode example.....	115
Figure 9-27: Tuning the feed-forward path in Tuning mode.....	115
Figure 9-28: Generate a model in Mission mode menu.....	116
Figure 9-29: Generate a reference model in Mission mode example.....	117
Figure 9-30: Tuning the feed-forward path in Mission mode.....	118
Figure 9-31: Import, export, and remove a model.....	118
Figure 9-32: Adjusting Feedback Gain Margin and Feedback Phase Margin in Configuration dialog.....	120
Figure 9-33: Amplitude phase plot of ANC Filter Designer window.....	124
Figure 9-34: Feed-forward path Configuration dialog.....	125
Figure 9-35: Automatic feed-back tuning using Automatic mode or Hybrid mode.....	127
Figure 9-36: Echo canceler path configuration dialog.....	129
Figure 9-37: Echo canceler filter spectrum.....	130
Figure 9-38: Echo canceler tuning example.....	131
Figure 9-39: Saving a ANC Filter Designer Snapshot.....	134
Figure 9-40: Snapshot status bar.....	135
Figure 9-41: Load/save ANC tuning parameters to an HTF file.....	135
Figure 9-42: Select UCID to read.....	136
Figure 9-43: Set UCID in HTF File Save Dialog.....	136
Figure 9-44: Open and Save ANC Workspaces dialog.....	137
Figure 9-45: Selecting an ANC mode to be written.....	138
Figure 9-46: Export IIR.....	139

Figure 9-47: Frontend Gains dialog.....	140
Figure 10-1: Setup: deploying ANC to production.....	142
Figure 10-2: ANC production calibration setup.....	146
Figure 10-3: ANC Write Fine Gain Delta.....	148
Figure 10-4: AT events sequence.....	153
Figure 10-5: Device State.....	154
Figure 10-6: Writing calibrated gain parameters.....	164
Figure 10-7: Reading from and writing to parameter storage.....	165
Figure 10-8: Reading from and writing to the HTF file.....	166
Figure 11-1: vailable earbuds in the GAIA application.....	171
Figure 11-2: Earbud settings.....	172
Figure 11-3: Earbud test results.....	173
Figure 11-4: Example of a good source file for wear status tuning.....	174
Figure 11-5: Example of a bad source file for wear status tuning.....	174
Figure 11-6: Example of the Power Ratio for a good source file.....	175
Figure 11-7: Example of the Power Ratio for a bad source file.....	175
Figure 11-8: Ambient level condition parameters.....	176
Figure 11-9: Threshold observations in QACT.....	176
Figure 11-10: Threshold observation using aancllogger.....	177
Figure 11-11: Clipping condition parameters.....	178
Figure 11-12: Clipping observations using aancllogger.....	179
Figure 11-13: ED observations using aancllogger.....	180
Figure 11-14: Earbud fit test threshold parameter.....	181
Figure 11-15: QACT example showing a good fit power ratio.....	181
Figure 11-16: Example power ratio plot.....	182
Figure 11-17: EFT example using aancllogger.....	183
Figure 11-18: GAIA Application: Audio curation Demo mode.....	186
Figure 11-19: 1-Mic detection levels.....	187
Figure 11-20: Wind intensity levels.....	189
Figure 11-21: Wind state.....	191
Figure 11-22: Enabling Auto Transparency feature.....	193

Figure 11-23: ATR Release time options.....	194
Figure 11-24: VAD power envelopes.....	195
Figure 11-25: VAD likelihood, triggers, and detections.....	196
Figure 11-26: ATR_VAD messaging state machine.....	197
Figure 11-27: AAH capability system block diagram.....	199
Figure 11-28: AAH algorithm block diagram with parameters.....	200
Figure 11-29: NoiseID state machine.....	208
Figure 11-30: NoiseID capability for Type-based classification from the logger tool.....	209
Figure 11-31: Weighting filter response curves.....	210
Figure 11-32: NoiseID capability for based classification from the logger tool.....	211

1 Qualcomm® Active Noise Cancellation for Earbuds Application

Qualcomm QCC517x, QCC307x, QCC518x, and QCC308x series chipsets support the Qualcomm ANC feature (ANC) for headsets and earbuds. The ANC operation is performed within a hardware acceleration block within the chipset. The Qualcomm Application Development kit (ADK) software builds in support for the ANC feature including software APIs to access the ANC hardware, Earbud Application examples, a tuning workflow to design ANC filters, and a production-line calibration example.

Topics include:

- [Supported chipsets](#)
- [ANC architecture and features](#)
- [Understanding microphone configurations](#)
- [ANC development process](#)
- [Configure ANC in the Earbuds Application](#)
- [Adaptive ANC and Adaptive Ambient tuning using Qualcomm Audio Calibration Tool](#)
- [Static ANC tuning using ANC Filter Designer](#)
- [Production-line calibration](#)
- [Adaptive ANC tuning use cases](#)
- [ANC hardware specification](#)
- [ANC stream APIs](#)

1.1 Supported chipsets

[Table 1-1](#) table lists the supported chipset platforms.

Table 1-1 Supported chipset platform

Chipset Platform			
QCC5120	QCC3040	QCC5141	QCC5181
QCC5121	QCC3044	QCC5144	QCC3081
QCC5124	QCC3046	QCC5151	QCC3083
QCC5125	QCC3042	QCC5171	QCC3084
QCC5126	QCC3050	QCC3071	QCC3086

Table 1-1 Supported chipset platform (cont.)

Chipset Platform			
QCC5127	QCC3056	QCC3072	QCC3091
			QCC3095
			QCC3096

2 ANC architecture and features

Qualcomm® QCC517x, QCC307x, QCC518x, and QCC308x series chipsets support the Active Noise Cancellation (ANC) feature for headsets and earbuds. The ANC operation is performed within a hardware acceleration block within the chipset. The Qualcomm Application Development kit (ADK) software builds in support for the ANC feature including software APIs to access the ANC hardware, Earbud Application examples, a tuning workflow to design ANC filters and a production line calibration example.

2.1 ANC architecture

The ANC architecture includes custom ANC hardware, audio APIs, Earbud Application, tuning tools, and production-line API with example scripts.

Figure 2-1 shows a top-level, simplified logical diagram of the ANC architecture.

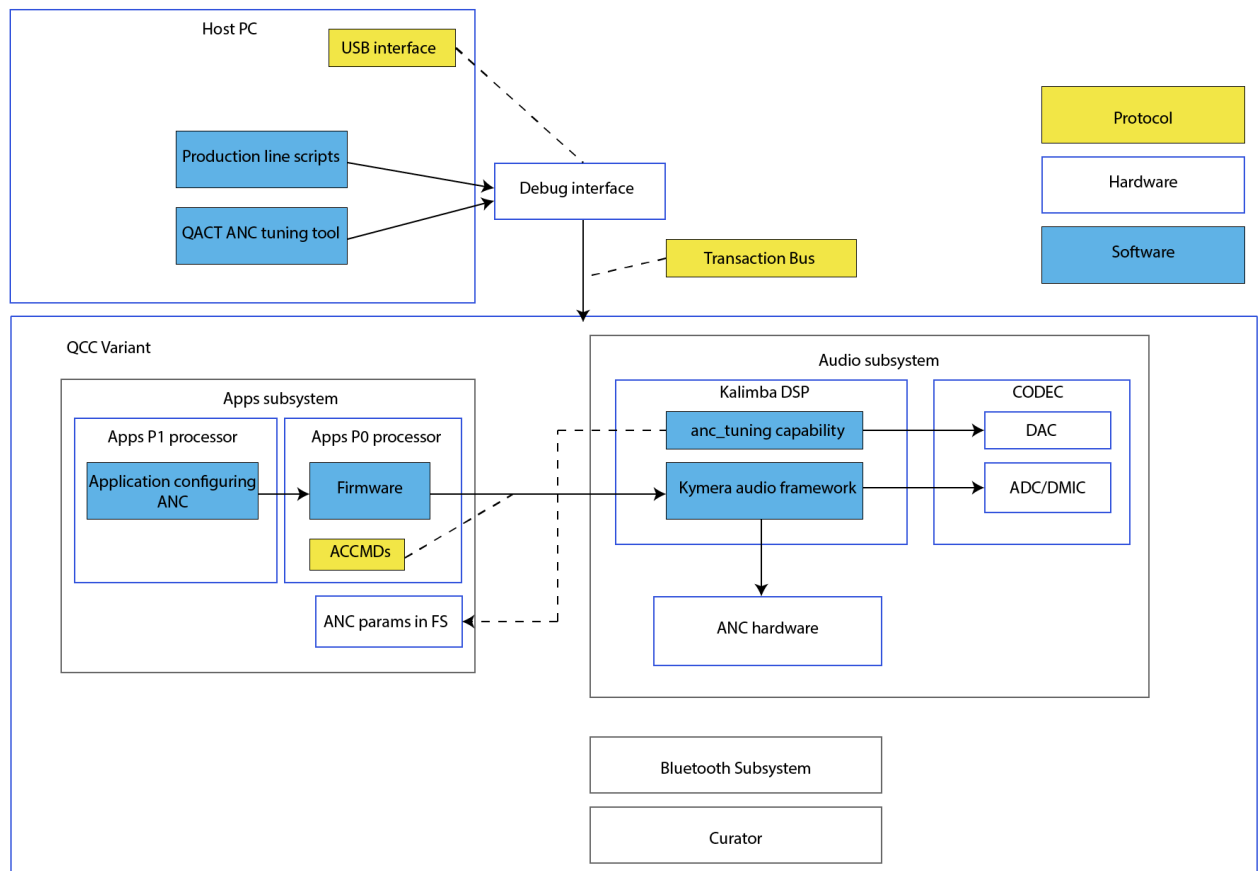


Figure 2-1 Top-level logical diagram of the ANC architecture

The ANC feature includes:

- Custom ANC hardware built into the codec within the chipset.
- ANC audio APIs for configuring the built-in ANC hardware of a device.
- ADK Earbud Application use cases that support ANC.
- Qualcomm Audio Calibration Tool (QACT™) ANC tuning workflow to design and tune ANC filters.
- ANC filter designer tool for tuning static ANC.
- Production line API and example scripts for calibrating ANC hardware in plastics.

2.2 ANC hardware block

The ANC hardware block has two instances *ANC0* and *ANC1* in a stereo configuration. The *ANC0* instance is associated with the left speaker. In an earbud configuration, each earbud uses *ANC0* associated with the speaker in that earbud. The *ANC1* instance is associated with the right speaker. In devices supporting Enhanced ANC, *ANC1* can also be tied in parallel to the same speaker as *ANC0*.

Table 2-1 describes the programmable paths associated with ANC hardware block.

Table 2-1 Programmable paths associated with ANC hardware block

Programmable paths	Associated with	Features
FFa: Filter Path A	Microphone	- Includes an IIR filter, fine and coarse gain control, low pass filter, and DC blocker. - Functions as a feedforward path in feed-forward ANC mode. - Functions as a feedback path in Feedback Mode and Hybrid Mode.
FFb: Filter Path B	Microphone	- Includes an IIR filter, fine and coarse gain control, low pass filter, small low pass filter and DC blocker. - Functions as a feedforward path in Hybrid ANC Mode. - Assumes that at least Filter path A is present in the system.
FB: Feedback compensation	Playback signal	- Includes an IIR filter, fine and coarse gain control, and low pass filter. - Applies an Echo Cancellor (EC) based on the acoustic path filter path A to preserve playback music or speech quality. - Assumes that at least Filter path A is present in the system and is performing the function of a feedback path.

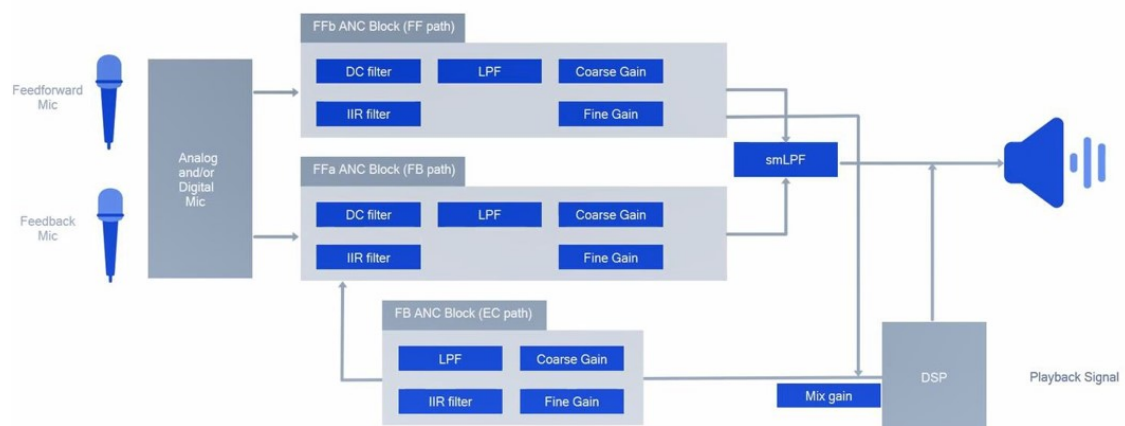


Figure 2-2 ANC hardware block

2.3 ANC modes

The ANC feature supports various use cases targeting stereo headset products and earbud products. Depending on the use case, the ANC feature may require up to six microphones, which must be set up when configuring the application.

For a maximal configuration of six microphones:

- Four microphones are used for ANC. These microphones further breakdown into two microphones per earpiece, which allow ANC to function in a hybrid mode using one external and one internal microphone per earpiece.
- Two microphones are used for characterizing the ANC audio paths during tuning. These microphones do not require physical microphones to be attached to the device. However, they require software microphone interfaces for tuning purposes. The application automatically configures these interfaces.

NOTE These interfaces are not required in the target product configuration running the application.

Simpler microphone configurations allow ANC to function in feed-forward or feedback modes.

Feed-forward ANC mode

In feed-forward (FF) mode, ANC takes the reference noise before the noise arrives at the actuator (speaker or receiver). It then computes reversed form of the noise (*anti-noise*) and plays the *anti-noise* through the speaker.

FF ANC is an ideal method to use when acoustic noise comes from one direction and can be captured earlier than it arrives at the ear drum of the user. FF systems can achieve high noise reduction in an ideal case. However, the performance of FF ANC can vary depending on the noise source direction.

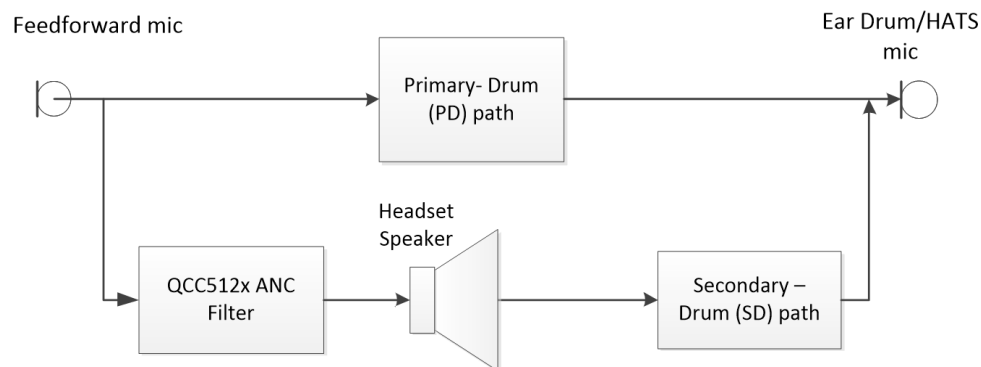


Figure 2-3 Feed-forward system

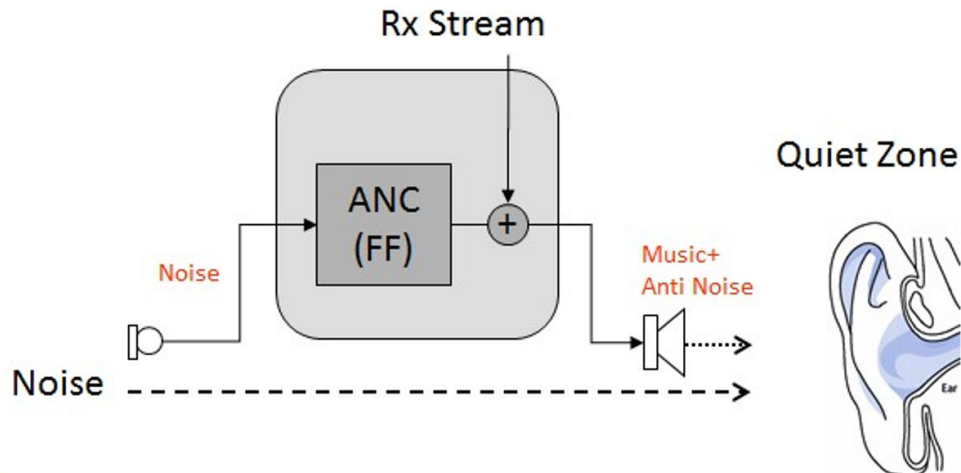


Figure 2-4 Feed-forward ANC

An FF ANC system can use FFa or FFb filters in one or both ANC instances. Because FFb is designed as a secondary mic path, it does not work without also enabling FFa. FFa is the preferred choice to minimize power consumption.

Feedback ANC mode

In feedback mode, ANC has a microphone that captures the remaining noises between the ear canal and the speaker (inside the quiet zone). The feedback algorithm processes these remaining noises to generate appropriate *anti-noise*.

Because the noise microphone also picks up music or voice signals sent to the Rx (speaker) path, Feedback ANC requires more complicated processing than FF ANC. In this case, the Feedback filter path is tuned to remove the residual music or voice signal from the microphone before processing (similar to an echo-canceller).

Because of the feedback nature of the configuration, instability can occur if careful tuning and adequate acoustic design is not in place. Integrating the microphone inside the quiet zone (between speaker and ear canal) might not only result in mechanical challenges but also affect stability and performance of the design.

Feedback ANC is typically more effective at reducing low frequency noise (below 500 Hz) and has less directional performance variation compared to feed-forward ANC.

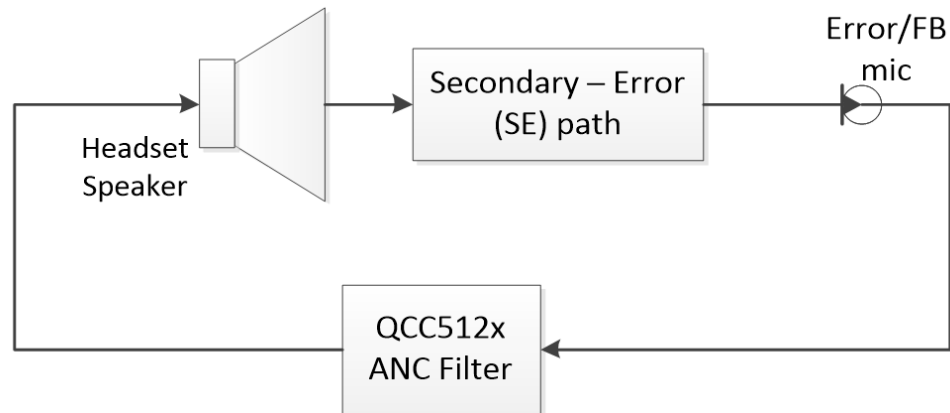


Figure 2-5 Feedback ANC system

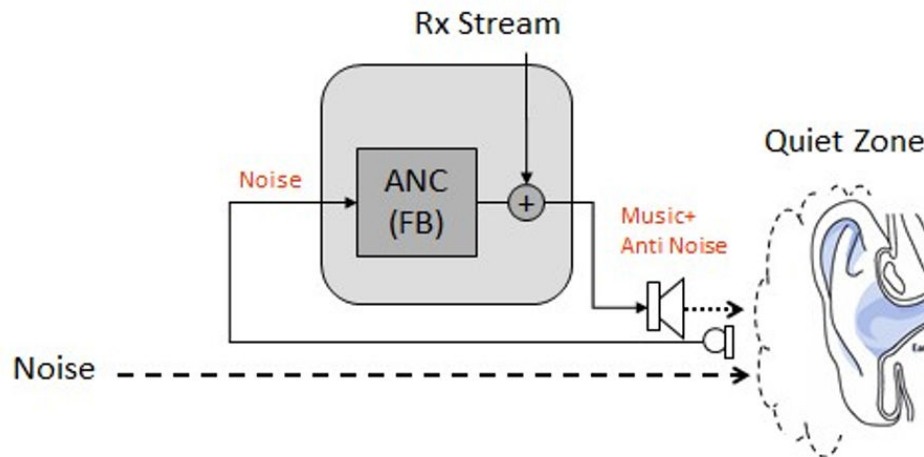


Figure 2-6 Feedback ANC

An FB ANC system can be designed with FFa and FB filters. FB is used for echo cancellation to remove music picked up by the feedback microphone.

Hybrid ANC mode

Hybrid ANC employs feed-forward and feedback paths that work together at the same time to preserve the advantages of the feed-forward and feedback noise cancellation ANC modes.

The Hybrid ANC requires a noise (FF Mic) and an error microphone (FB Mic) per channel to capture the reference noise before the actuator (speaker or receiver) and the remaining noises between ear canal and the speaker (inside the quiet zone). Independently, the signal from the noise microphone is processed by the FF path and the signal from the error microphone is processed by the FB path.

A hybrid ANC system can be designed with:

- FFb filters for feed-forward section.
- FFa and FB filters for the feedback section.

Enhanced ANC mode

Enhanced ANC (eANC) feature uses parallel topology configurations to realize filter shapes where increased complexity is required. In this configuration, both hardware ANC instances along with both DAC channel (see note) are used for both feed-forward and/or feedback ANC paths. The filter configuration of the ANC hardware can be different for each ANC instance, which may enable implementation of more complex composite filters. Multiplexing the microphone data to both ANC instances further helps earbuds to improve overall noise cancellation experience.

The Enhanced ANC mode can be used instead of single topology Hybrid ANC, where a complex ANC filter shape is needed. The complex target frequency response needs multiple inflection points to get good ANC attenuation and eANC (in parallel) serves this purpose.

Enhanced ANC (eANC) can be used along with Adaptive ANC mode as well. For simple ANC filter shapes, the ANC performance may be very similar with single filter and parallel filter topologies.

NOTE The unused DAC channel in mono earbud application for enhanced ANC is used for echo cancellation in second ANC instance (ANC1).

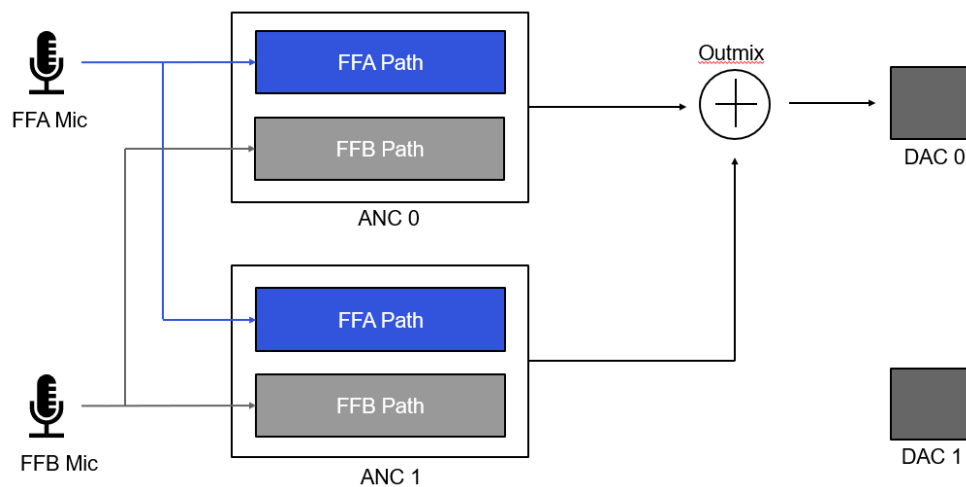


Figure 2-7 Enhanced ANC mode

Adaptive ANC mode

Adaptive ANC reduces the impact of earbud fit changes on ANC performance by automatically modifying the FF gain as fit changes. In Adaptive ANC mode, the DSP capability uses external (feed-forward) and internal (error) microphone signals to adjust the gain in the feed-forward path. The internal microphone may also do feedback ANC, but this is not necessary for adaptation.

The adaptive gain algorithm applies bandpass filters to both signals, then filters the external microphone through a model of the Plant-Controller that is active on the ANC hardware. It then uses an FxLMS routine to calculate the required change in gain.

Additional monitoring blocks pause the adaptation to ensure that it performs correctly across different noise environments:

- Presence of speech (Energy Detectors)
- Presence of self-speech (Energy Detectors)

Signal overload, for example from tapping or rubbing the device, from incorrectly influencing the calculation (Clipping Detectors, Saturation Detectors).

NOTE In this use-case, when the DAC floor noise is dominating the ambient noise, the earbuds automatically go into the ultra quiet DAC mode. When the surrounding ambient noise is loud, the earbuds come out of the ultra-quiet mode automatically.

Adaptive ANC in single mode, where only one ANC instance is used. The adaptive gain algorithm applies bandpass filters to both signals, then filters the external microphone through a model of the Plant-Controller that is active on the ANC hardware. It then uses an FxLMS routine to calculate the required change in gain.

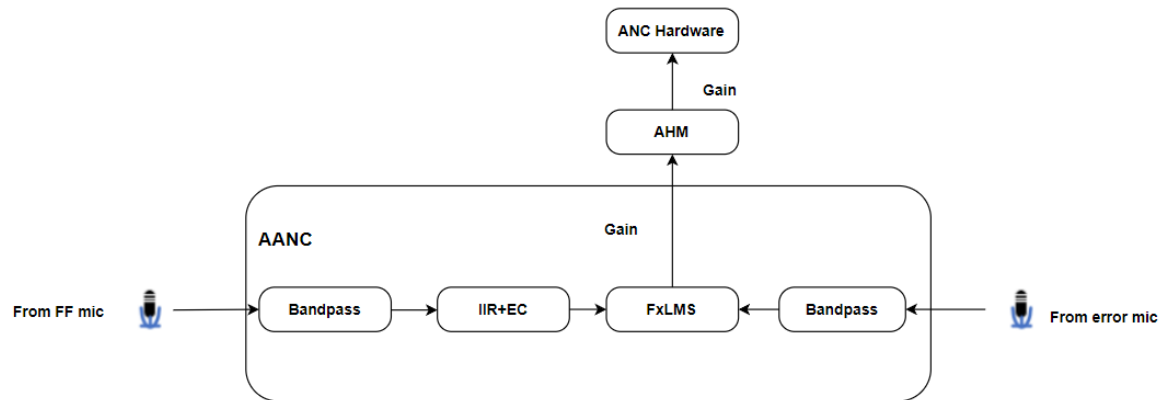


Figure 2-8 Adaptive ANC in single mode

Adaptive ANC in Enhanced mode utilizes both hardware ANC instances. The adaptive gain algorithm applies bandpass filters on both signals and processes the external microphone signal through a model of the active Plant-Controller on each ANC hardware instances separately. Each ANC instance may have a distinct filter configuration, allowing for the implementation of complex composite filters. An FxLMS routine is then used to determine the necessary gain adjustment, which is uniformly applied across both ANC instances

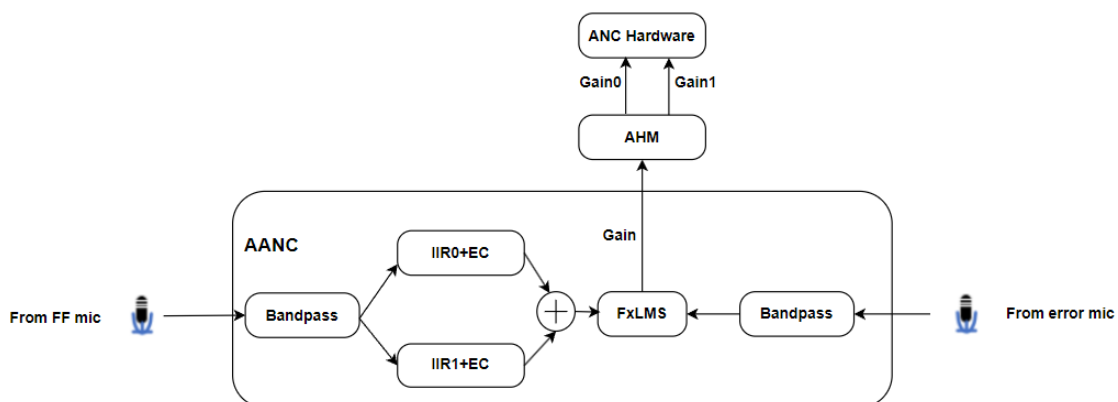


Figure 2-9 Adaptive ANC in Enhanced mode

Adaptive ANC in Enhanced Dual mode utilizes both hardware ANC instances. The adaptive gain algorithm applies bandpass filters on both signals and processes the external microphone signal through a model of the active Plant-Controller on each ANC hardware instances separately. Each ANC

instance may have a distinct filter configuration, allowing for the implementation of complex composite filters. An FxLMS routine is then used to determine the necessary gain adjustment for each instance separately and applied to the corresponding ANC instance.

Adaptive ANC in Enhanced Dual mode utilizes both hardware ANC instances. The adaptive gain algorithm applies bandpass filters on both signals and processes the external microphone signal through a model of the active Plant-Controller on each ANC hardware instances separately. Each ANC instance may have a distinct filter configuration, allowing for the implementation of complex composite filters. An FxLMS routine is then used to determine the necessary gain adjustment for each instance separately and applied to the corresponding ANC instance.

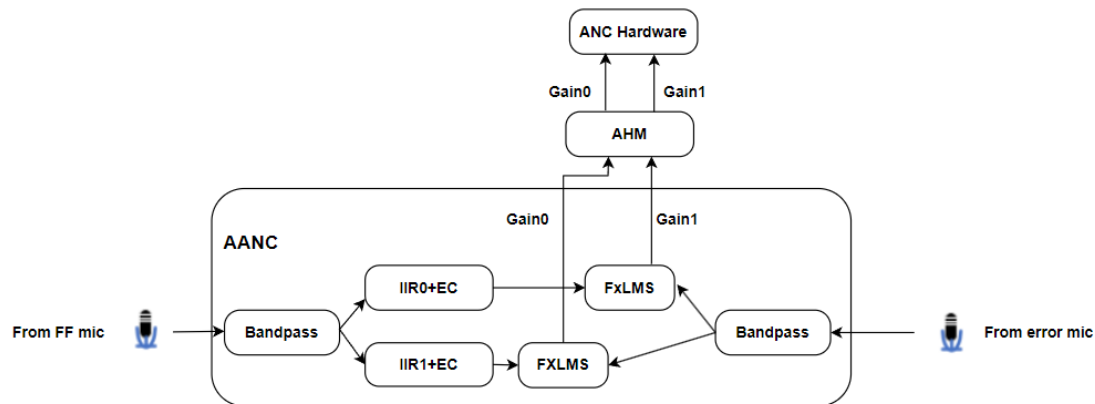


Figure 2-10 Adaptive ANC in Enhanced Dual mode

World Volume/Leak-through ANC mode

The World Volume mode of ANC is a leak-through mode of ANC where Earbud Application uses ANC feed-forward mic to give awareness of the ambient environment and amplifies/attenuates sound by end user control either through GAIA mobile application or using dedicated button events.

The World volume mode can be used with both Static ANC and Adaptive ANC.

World Volume mode in Static ANC

The World Volume mode in Static ANC configuration is also known as ANC Ambient or ANC Leakthrough mode. To configure this mode, see *Defining roles for different ANC modes (Applicable from ADK 21.1 onwards)*.

World volume mode in Adaptive ANC

The World Volume mode in Adaptive ANC configuration is also known as Adaptive Ambient mode or Adaptive Leakthrough mode. Adaptive Ambient ANC mode makes quieter sounds sound louder without hurting ears. For configuring this mode, see *Defining roles for different ANC modes (Applicable from ADK 21.1 onwards)*.

2.4 ANC use cases

The ADK Application instantiates different use case graphs within the Qualcomm® Kymera™ audio framework for different ANC use cases.

The ADK Application is an implementation of an audio rendering device supporting Bluetooth® technologies. It is an on-chip application designed to run explicitly on supported Qualcomm chipset. The source code provided is modifiable.

The Application supports the following ANC use cases:

- ANC with music playback (Bluetooth, USB input, I2S input)
- ANC with voice applications (for example, Qualcomm® cVc processing)
- ANC standalone (no audio playback)
- ANC tuning (USB play and record)

Static ANC tuning use case

For the ANC tuning use case, the Application instantiates a USB play and record graph with a `download_anc_tuning` capability in the Audio subsystem. This use case graph configures the ANC hardware block and setup routing for recording various acoustic paths required to design an ANC filter.

In the tuning use case, the ANC hardware is controlled by the `download_anc_tuning` capability. Use the tuning use case along with the QACT ANC tuning tool. The output of the tuning process is a set of parameters to be saved to the device in the form of an HTF file.

For a complete Earbud Application implementation, and information about how the ANC tuning use case is setup and configured, see `<ADK path>\adk\src\domains\audio\kymera\kymera_anc.c`.

NOTE The tuning use case is intended for laboratory use and is not intended for production use cases.

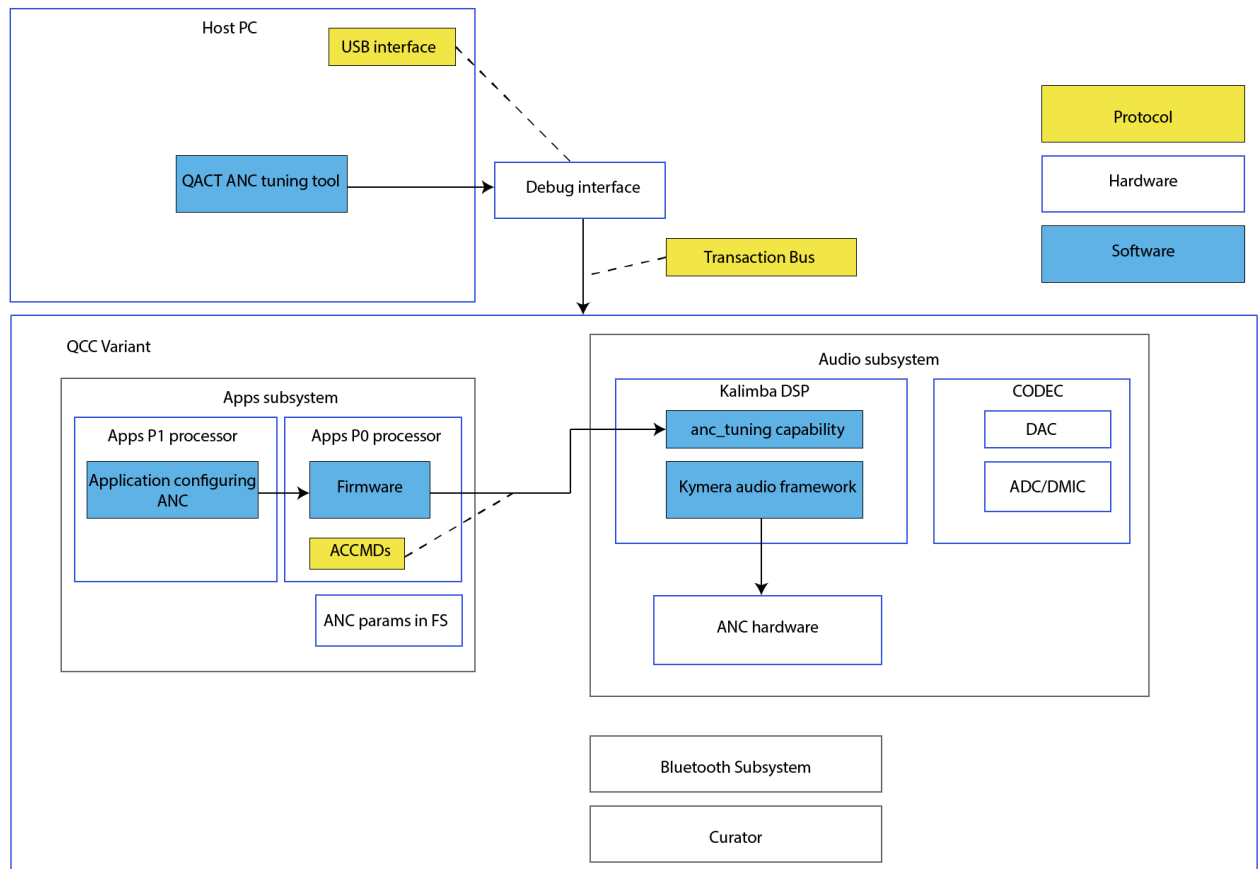


Figure 2-11 Logical diagram of ANC tuning use case

Static ANC Run Mode use case

ANC run mode use cases are instantiated by the end user. ANC parameters are loaded into hardware, and noise cancellation occurs alongside music streaming or voice playback.

ANC run mode use cases in the Application can be instantiated by an appropriate use case graph within the Kymera audio framework. Here, the ANC block is set up by directly configuring the audio endpoints (ADC/DAC) associated with ANC and through functions such as:

`AudioAncFilterIirSet`, `AudioAncStreamEnable`, and `AudioAncFilterLpfSet`.

For a complete application implementation of how the ANC hardware is set up and configured, see `<ADK path>\apps\libs\anc\CONFIG_HYDRACORE` or `<ADK path>\adk\src\libs\anc\`.

ANC leak-through mode use case

ANC leak-through mode use cases are instantiated by the end user. ANC parameters are loaded into hardware, and microphone signals are passed using the ANC hardware, which get mixed alongside music streaming or voice playback. Apart from attenuating or amplifying the microphone, no other modifications are made to the original signal.

ANC leakthrough mode use cases in the Application can be instantiated by an appropriate use case graph within the Kymera audio framework. Here, the ANC block is set up by directly configuring the audio endpoints (ADC/DAC) associated with ANC and through functions such as:

- `AudioAncFilterIirSet`
- `AudioAncStreamEnable`
- `AudioAncFilterLpfSet`

Qualcomm Adaptive ANC Run Mode use case

Qualcomm Adaptive ANC reduces the impact of earbud fit changes on ANC performance by automatically modifying the FF gain as fit changes. In Qualcomm Adaptive ANC mode, the DSP capability uses external (feed-forward) and internal (error) microphone signals to adjust the gain in the feed-forward path. The internal microphone may also do feedback ANC, but this is not necessary for adaptation. For more information about Adaptive ANC mode, see [Adaptive ANC mode](#).

Adaptive ANC Tuning Mode use case

The Adaptive ANC tuning mode is designed to provide insight and understanding when developing Adaptive ANC algorithms. It instantiates a graph that records either the feed-forward and error microphone signals or the bandpass-filtered versions (depending on the `AANC_DEBUG` parameter, see the *Kymera Capability Library User Guide*). These can be used in development to analyze algorithm behavior offline, for example in `kalsim`.

Parameter storage in ANC tuning use case

The output of the ANC tuning use case, described in [Static ANC tuning use case](#), is a parameter file written to the file system on chip. The ANC parameter information is used as an input to the ANC processing use cases described in [Static ANC Run Mode use case](#).

For a complete Application implementation of how the ANC tuning parameters are transferred to ANC processing use case, see `<ADK path>\apps\libs\anc\CONFIG_HYDRACORE` or `<ADK path>\adk\src\libs\anc\`.

The QACT ANC tuning tool provides the ability to save ANC parameters as a `*.htf` file. It is possible to later build the file into a file system, or download directly to the device on Audio subsystem shutdown using the `Write to persistence` feature in QACT. In both cases, the persistent store audio key (See *PsReadAudioKey function prototype*, below) that is used for copying data is a fixed number that is derived from the following:

```
#define MAP_TO_PSKEYID(capid, ucid, sbid)

(((capid) & CAPID_PS_MASK) << CAPID_PS_SHIFT) | \
((ucid) & UCID_PS_MASK) << UCID_PS_SHIFT)      | \
(((sbid) & SBID_PS_MASK) << SBID_PS_SHIFT))
```

Where, CAPID is the capability ID for the `download_anc_tuning` capability – that is, `0x4082`. Each use case ID can represent a unique filter parameter set. In the Application UCID = 0 is used for ANC

mode_1, UCID = 1 is used for mode_2 and so on, until UCID = 9, which is used for mode_10. Each UCID (ranging from 0 to 9) can set a custom ANC filter configuration that plays active noise from the speaker. SBID = 0 for all ANC tuning use-cases at present. The QACT ANC tuning tool allows multiple UCIDs to be created for different filter configurations.

When using Adaptive ANC, the Adaptive ANC capability along with the Static ANC hardware needs to be tuned. The starting capability id for Adaptive ANC is 0x204F80. The UCID for adaptive ANC refers to adaptive ANC modes where UCID = 0 corresponds to Adaptive ANC mode 1. The adaptive ANC capability ucid and ANC tuning capability ucid works together to form different adaptive ANC modes, for example Adaptive ANC mode 0 functionality is achieved by download_anc_tuning UCID- 0 and download_aanc UCID-0 configuration. Similarly the Adaptive ANC mode 1 is defined by download_anc_tuning UCID-1 and download_aanc UCID-1 capability configuration.

When tuning ANC, the `AEC_REF` capability must be tuned. When the `cvc` microphones are shared with the ANC microphones, the microphone gains for `AEC_REF` and ANC tuning capability must be same.

In QCC517x and QCC307x devices, the Adaptive ANC noise cancellation mode requires tuning the ANC Hardware manager (AHM), Howling Control and Gain Reduction (HCGR), and Adaptive ANC v2 (AANCv2) capabilities. For Adaptive leak-through use case, the ANC Compander capability must be tuned.

The Application provides a default *.htf file built into the `user_ps_filesystem`. This file enables the users to run an ANC processing use case with a pass through configuration without having to design and tune ANC. An ANC pass-through configuration sets up the IIR filters within the ANC block to send microphone data straight through to the speaker without filtering and unity gain within the ANC hardware.

The following example shows a pass-through configuration *.htf file for a feed-forward ANC system.

```
file = audio
```

```
# Default feedforward ANC_mode_1 with unity gain and flat filter for FFa path
on left and right
```

```
# PSID=0x204100, capID=0x4082, UCID=0x0000
```

[illegible]

```

00 00 00 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ]

```

In this example, 0x204100 is the PSID derived from `MAP_TO_PSKEYID(capid, ucid, sbid)`. The values are byte-swapped. The payload containing the ANC parameters is displayed after the initial header information 01 10 00 00 B8 00.

The ANC parameters in processing use cases are read from `PsReadAudioKey` using:

```

uint16 PsReadAudioKey(uint32 key, uint16 * buffer, uint16 words, uint16
offset, uint16 * key_length_addr);

```

2.5 Adaptive ANC capability

Adaptive ANC is an additional capability available under licence for True Wireless Earbuds. For more information about Adaptive ANC mode, see [Adaptive ANC mode](#).

The Adaptive ANC capability is designed to be tuned in run mode through QACT and the `aanclogger` tool.

2.6 Support for Custom Adaptive ANC

The Qualcomm Adaptive ANC capability (Adaptive ANC) can be extended using the source code and tools provided in the ADK.

Adaptive ANC playpen

The Adaptive ANC capability source is provided with the ADK. The downloadable project contains files, which are listed in [Table 2-2](#).

Table 2-2 Adaptive ANC playpen files

Files	Description
■ <code>aanc_cap.c</code> ■ <code>aanc_cap.h</code>	Core capability wrapper with standard Kymera API functions in addition to the following key functions: ■ <code>update_gain</code>: Uses the stream API to update the ANC hardware. ■ <code>process_event</code>: Deals with event messaging, for example communicating the quiet mode condition to the application. ■ <code>process_event</code>: Deals with event messaging, for example communicating the quiet mode condition to the application. ■ <code>set_static_gain</code>: Message handler to receive gains used for static mode.
■ <code>aanc_proc.c</code> ■ <code>aanc_proc.h</code> ■ <code>aanc_proc_asm.asm</code>	Adaptive ANC algorithm specific processing. This is the best place to incorporate custom algorithms, and involves the following four key functions:

Table 2-2 Adaptive ANC playpen files (cont.)

Files	Description
	▪ <code>create, destroy</code>: (De)allocates custom data structures. ▪ <code>initialize</code>: Run-time initialization. ▪ <code>process_data</code>: Algorithm functionality.
<code>aanc_defs.h</code>	Common definitions across the AANC capability: frame size, frame rate, etc.
<code>aanc.h</code>	Standard capability header file.
<code>aanc_gen_defs.c</code>	Default parameter value.

The `process_data` function within the capability wrapper performs various checks on the state of the operator such as:

- In Ear status is valid.
- ANC hardware clocks are valid.
- Data is available to process.

The `process_data` function then calls into the functions provided by `aanc_proc` for initialization and algorithm processing.

The `process_data` function in `aanc_proc` performs the following activities:

- Copies input data to internal buffers.
- Runs clipping detection on all inputs. Returns early if clipping detected.
- Runs energy detection on all inputs (using the `ed100` private library) and sets flags accordingly.
- Runs self-speech detection (using the `ed100` private library) and sets flags accordingly.
- *<Custom algorithm>*: Sets flags as specified.
- Runs the adaptive gain algorithm (using the `fxlms100` private library) and tests flags and runs accordingly.
- Copies data to output buffers (if applicable).

The core algorithm functionality can be adapted in this function. An example is listed above (*<Custom algorithm>*) for a place to add, for example, a custom algorithm that operates on the input data and controls whether gain adaptation occurs or not. Another example would be replacing the adaptive gain algorithm with a custom algorithm.

After calculating the gain, the capability wrapper `process_data` function performs the following activities:

- Checks the flags to determine any mode override behavior.
- Checks the operator mode to determine the update values for the final gain.
- Calls the `update_gain` function to write these values, either to hardware or in KSE, as an unsolicited message that can be handled within the simulator to enable closed loop simulation.

To customize the functionality, additional modifications can be made at the flag checkpoint or in the operator mode state machine to customize functionality.

DSP simulation environment for Adaptive ANC

The Adaptive ANC capability can be run in KSE with a python filter operator that mimics the ANC hardware to provide for a closed loop simulation. An example is provided that contains the following files in the KSE workspace (available at <audio package path>\kse\workspace):

- `config\aaanc.cfg.json` file: This is a JSON configuration file that describes the graph. [Figure 2-12](#) demonstrates the input noise signal being split between two paths. The external microphone receives the signal directly. The internal microphone is filtered using the following three FIR filters, which are set to unity gain for simulation purposes.
 - Filter 2 models the ANC hardware and is gain-controlled using an unsolicited message from the AANC capability.
 - Filter 3 models the transfer function between the ANC hardware and internal microphone (S model).
 - Filter 4 models the transfer function between the external environment and the internal microphone (P model).

[Figure 2-13](#) is same as parallel filter except the internal microphone is filtered one extra FIR filter.

- Filter 2 models the ANC hardware instance 0 and is gain-controlled using an unsolicited message from the AANC capability.
 - Filter 7 models the ANC hardware instance 1 and is gain-controlled using an unsolicited message from the AANC capability.
- `resource\white_1c_16000_16b_3s.wav` file: White noise input file.
 - `script\kalsim\aaanc.py` file: Simple control script that runs when KSE is launched. This creates the graph, associates the callback to control gain, and plays the graph.

When the simulation is run, the unsolicited messages are printed to the screen and show gain convergence at 127 for the default project and configuration. The output wavefiles available in the

workspace\tmp folder display the corresponding converged attenuation on the signal at the internal microphone.

NOTE For the dual filter configuration, the gain convergence will be different for both the ANC instances.

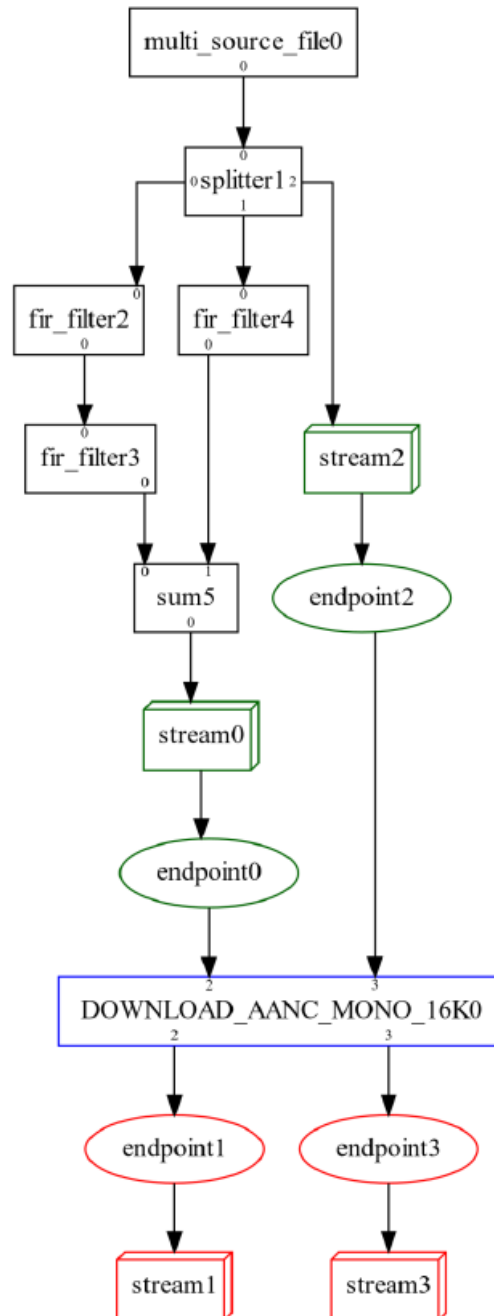


Figure 2-12 KSE simulation graph for Enhanced ANC

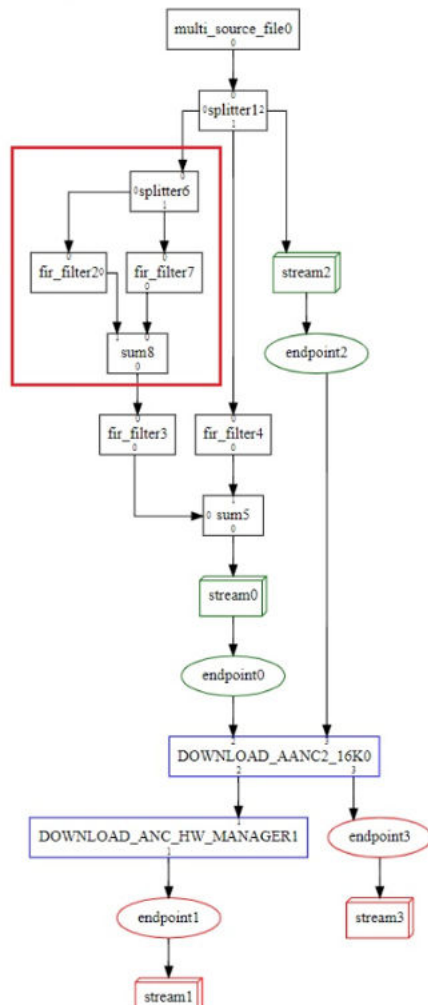


Figure 2-13 KSE simulation graph for dual filter topology

Launch KSE for ANC simulation

To launch KSE with support for a simple ANC simulation:

- ADK 20.3 and onwards: From MDE select **kse** as the target for the **download_aanc project**, and then deploy.
- ADK 20.2:
 - a. Copy the workspace files provided to the KSE workspace in the <audio package path> \tools\kse\workspace location.
 - b. Install python libraries `numpy` and `scipy` into the KSE python environment.
 - c. Edit the project settings for KSE (**Build Settings > User**).
 - d. Modify `KALSIM_SCRIPTS` and replace `download.py` with `aanc.py`.
 - e. Deploy.

DSP data logging environment for Adaptive ANC

`aanclogger` is a tool that uses ACAT to continuously poll the operator state and plot information to assist in understanding the algorithm behavior.

For detailed instructions about how to install, run, and modify the tool configuration to suit your development requirements, see the *Qualcomm Kymera Tools User Guide*. In ADK 20.2, the documentation is provided with the tool as a collection of HTML files.

To configure the `aanclogger` tool:

1. Install the python wheel into a python environment.
2. Setup Adaptive ANC on your device and note the USB Debug port (for example, `tc/usb2tc/100`).
3. Launch the tool.

Capability extensions with QACT

Parameters and statistics can be modified using the Capability Designer feature in QACT. There are changes required in two places, in the QACT and capability:

- To make changes in QACT:
 - a. Open the workspace file in offline mode.
 - b. Launch the Capability Designer by navigating to **Tools > Capability Designer**.
 - c. Select the capability.
 - d. Modify the parameters and statistics, as required.
- To make changes in the capability:
 - a. Modify the `aanc_gen_c.h` file as required to update the parameter and statistic definitions.

- NOTE**
- If the `aanc_gen_c.h` file is not present in the `download_aanc` project, copy it from `<kalimba/kymera/output/streplus_rom_release/gen/aanc>` to `<kalimba/kymera/capabilities/aanc>`, and then add it to the project.
 - Updates to the parameters require a corresponding update to the default parameters in the `aanc_gen_defs.cfile`.

2.6.1 Creating custom Adaptive ANC capability

Qualcomm ADK provides various tools, including an example Adaptive ANC playpen capability, a DSP simulation environment, and a DSP data logging environment, for developing custom Adaptive ANC capability.

Adaptive ANC playpen

An example capability, Adaptive ANC Lite, is provided for customers creating their own Adaptive ANC (AANC) capability. The AANC lite capability source is provided with the ADK. The downloadable project includes the following files:

- `aanc_lite_cap.c`, `aanc_lite_cap.h`: Core capability wrapper with standard Kymera API functions. The wrapper includes `aanc_lite_update_gain`, which demonstrates how to update the ANC hardware.
- `aanc_lite_defs.h`: Common definitions across the AANC capability including the frame size, frame rate, and so on.
- `aanc_lite.h`: Standard capability header file.
- `aanc_lite_gen_defs.c`: Default parameter values.

The `process_data` function within the capability wrapper performs the following activities:

- Checks if data is available to process.
- Checks the operator mode.
- Copies input data to output buffers (if applicable).

The `process_data` function also provides an example of ANC gain being updated using a simple gain ramp. This function can be replaced by a custom adaptive ANC algorithm as needed. It then calls the `update_gain` function to write the updated gain value (either to hardware or in KSE as an unsolicited message).

DSP simulation environment for Adaptive ANC

See [DSP simulation environment for Adaptive ANC](#). Note that the JSON configuration file is `aanc_lite.cfg.json`, to accommodate a different capability ID and terminal connections.

DSP data logging environment for Adaptive ANC

For more information, see [Support for Custom Adaptive ANC](#).

To represent the data structures available in the custom capability, the variable names that are plotted must be modified from the default JSON configuration.

Capability extension with QACT

See [Capability extensions with QACT](#).

3 ANC hardware specification

The ANC hardware layout, specification, and hardware components on the QCC517x, QCC307x, QCC518x, and QCC308x chipset is shown in [Figure 3-1](#).

Each chipset has two ANC hardware instances:

- ANC instance 0: Left channel ANC hardware
- ANC instance 1: Right channel ANC hardware

NOTE In mono use case, both the ANC instances are used for the left channel to maximize ANC performance. Each ANC instance has three filter paths:

- FFa: Filter path A
- FFb: Filter path B
- FB/EC: Echo canceller filter path

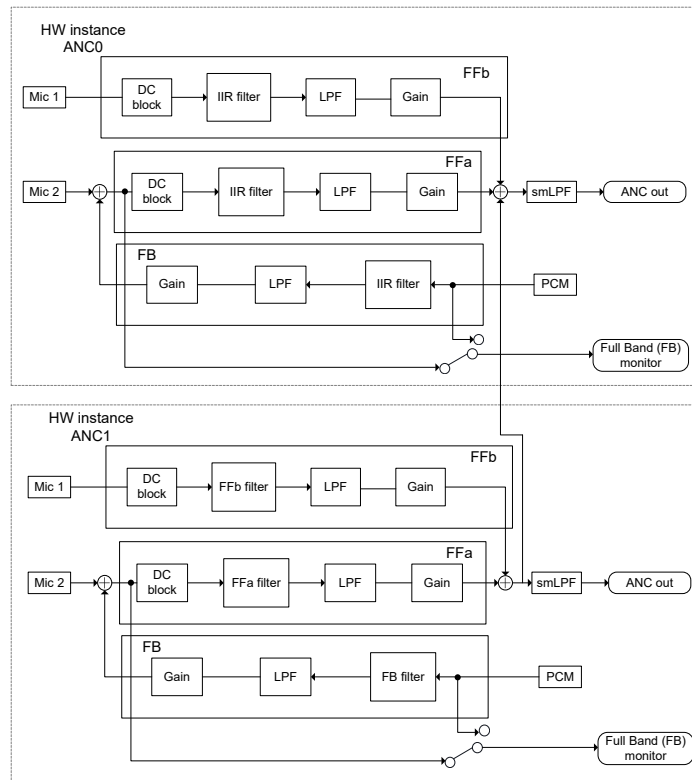


Figure 3-1 Enhanced ANC hardware and signal routing without RxMix

3.1 ANC filter parameters

The parameters (DC filter, smLPF, LPF) for each block can be selected from the drop down menu available in the filter designer tool. The filter coefficients and gains can also be given as input from the filter designer tool. This section provides the specifications for each of the tunable block in the QCC517x ANC architecture.

DC blocking filter

QCC517x has the DC filter to block the DC from microphones connected to ANC hardware. The DC filter can be disabled or set to a specific cut-off values listed in [Table 3-1](#).

Table 3-1 QCC517x DC filter cut-off frequencies for the recommended ANC sampling rates

32 kHz sampling rate (Hz)	64 kHz sampling rate (Hz)
160.4	320.8
79.55	159.1
39.65	79.3
19.8	39.6
9.9	19.8
4.95	9.9
2.45	4.9
1.25	2.5
0.6	1.2
0.3	0.6
0.2	0.3
0.1	0.2

[Table 3-1](#)

IIR coefficients

Each ANC instance contains three programmable IIR filters: FFa, FFb, and FB.

The coefficients are organized as eight denominator coefficients, followed by nine numerator coefficients in a single polynomial (eight zeros and eight poles), as described by the following equations.

In QCC517x, QCC307x, QCC518x, and QCC308x devices, the number of coefficients is increased from 15 to 17, which are 32 bits wide, represented in a Q8.24 format (includes sign bit). The audio control band is from 0 KHz to 32 KHz.

$$H(z) = \frac{H(z)}{P(z)} = k \frac{(z - z_1)(z - z_2) \dots (z - z_n)}{(z - p_1)(z - p_2) \dots (z - p_n)}$$

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_1 + b_2 z^{-1} \dots + b_{n-1} z^{-n} + b_n z^{-n-1}}{a_1 + a_2 z^{-1} \dots + a_{m-1} z^{-m} + a_m z^{-m-1}}$$

Figure 3-2 IIR coefficient equations

LPF

The Low Pass Filter (LPF) is available in each ANC path to control the audio band noise, to overcome the instability and to reduce noise overshoot. The cutoff frequency can be chosen per path (FFa or FFb or FB).

Table 3-2 QCC517x low pass filter cut-off frequencies for the recommended ANC sampling rates

32 kHz sampling rate	64 kHz sampling rate
732576	1465152
340034	680068
164346	328692
122254	244508
80848	161696
60394	120788
40102	80204
30018	60036
19972	39944
14964	29928
12464	24928
9966	19932
7472	14944
4978	9956
3732	7464
2488	4976
1866	3732
1244	2488
933	1866
622	1244

Gains

ANC hardware gains for each filter path has two parts:

- Coarse grains: Gain block with larger gain step size. Range is from -48 to 36dB with 6dB step size.
- Fine gains: Gain block with smaller step size.

The fine gain value is calculated using the formula:

$$\text{gaindB} = 20 * \text{math.log10}(\text{gain}/128)$$

Where gain is a value between 0 and 255. For example, some fine gain values are shown in [Table 3-3](#).

Table 3-3 Example fine gains

Gain parameter	Gain
255	6 dB
128	0 dB
64	-6 dB
0	-Inf dB (mute)

The default *.htf file parameters in the Sink Application sets fine gain to a value of 128 (0 dB). The total gain for each filter path within the ANC block is the sum of the coarse gain and fine gain.

When the filter designer tool is used, the gain displayed for each path is a combination of coarse and fine gain.

During production line calibration, fine gain is adjusted to compensate for the variation in electroacoustic component sensitivity.

smLPF

This block provides an extra low pass filtering stage applied on the output sum of FFa and FFb filters for a given ANC instance.

The cut-off frequency in Hz for smLPF:

- 882542
- 366288
- 170017
- 82173
- 40424
- 20051
- 9986
- 4983

NOTE On QCC307x and QCC517x devices, it is recommended to disable the smLPF (smoothing low-pass filter) to avoid adding noise to the audio signal. This is due to a known issue with the smLPF that causes it to introduce noise. If a low-pass filter is needed, it can be designed in the filter paths instead of using the smLPF.

ADC/DAC gains

The ADC and DAC gain parameters control the Codec gains within the CDA device. The user can enter a value between -90 dB and 90 dB. For more information on codec gain, see *ADK 6.x Audio API Design Guide*.

It is important that in mic sharing use cases (for example, with cVc voice processing), these gain values need to be identical in all use cases. The recommended approach to calibrating gains is to use the gain and shift values within the ANC hardware block.

NOTE Because of the nature of the ANC hardware, only the analog portion of codec gains effect ANC signals. For of analog mic gains, the valid range is 0 dB to 39 dB in steps of 3 dB. For digital mics, the codec gains have no effect.

Rxmix

An additional Rxmix/Selfmix path with gains are available in QCC517x, QCC307x, QCC518x, and QCC308x chipsets. This path is used to ensure that the feedback ANC does not attenuate ambient signals in leak-through mode.

The functionality of Rxmix in leak-through mode is similar to that of EC path for feedback ANC mode.

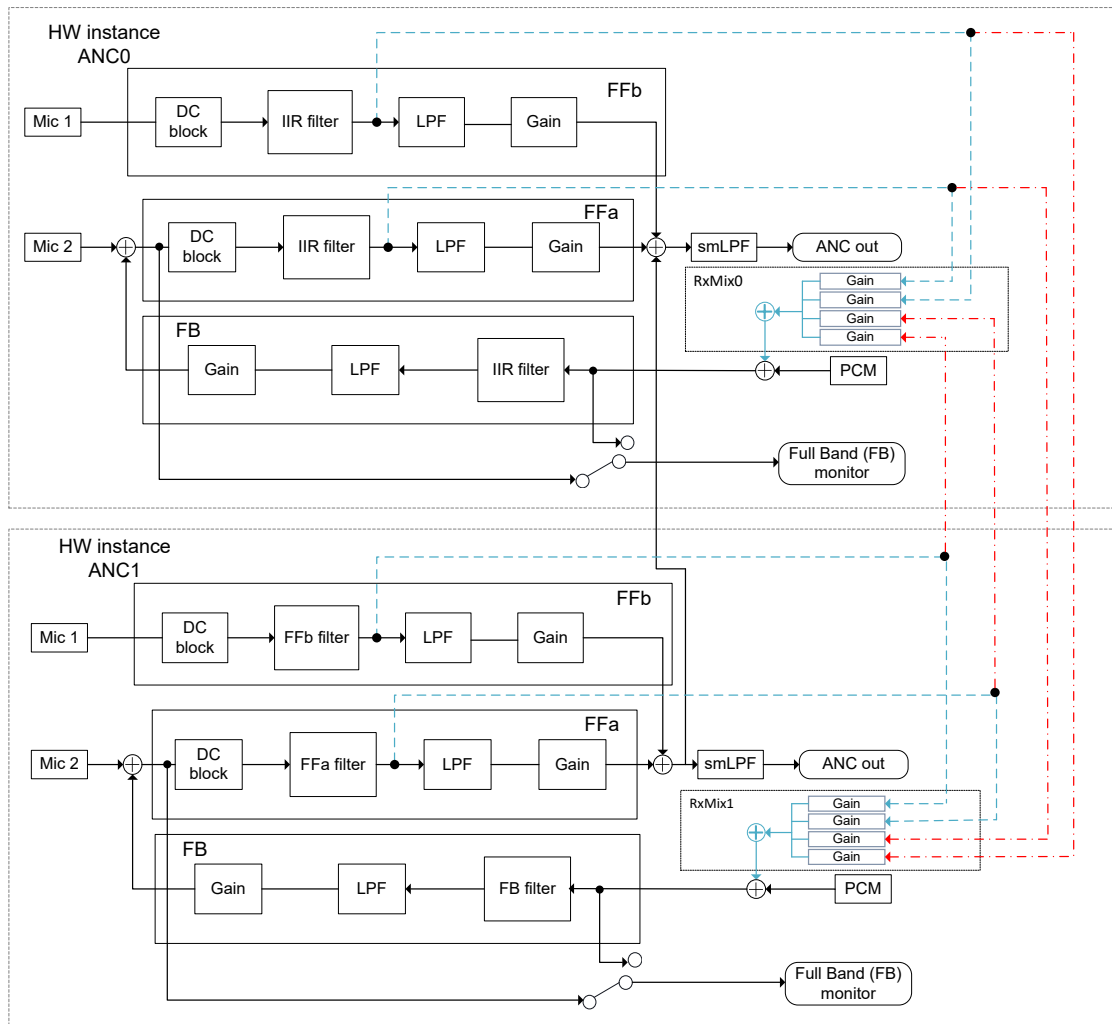


Figure 3-3 Enhanced ANC hardware and signal routing with RxMix

3.2 ANC full-band monitor mux

The ANC tuning capability, besides being a parameter interface for the ANC hardware block, is also an audio router. The ANC tuning capability has eight inputs (sinks) and four outputs (sources). The override controls allow the user to select any input to output routing for each of the four sources.

The inputs can be:

- USB input
- Upto four ANC microphones
- FB monitor taps
- Silence

FB monitor

The ANC hardware block provides monitoring points to stream the output of either the FFa or FB filters. If FB_MON is chosen as the input for one of the outputs, then extra controls are provided to choose if the FB monitor is associated with an FFa or an FB filter.

During ANC tuning, the mux routes signals for recording. Drop-down menus are available to control which audio signals are routed to and from USB.

The sources available are:

- USB L or R playback
- ANC microphones connected to FFa and FFb on each ANC instance
- FB_MON monitor outputs from each ANC instance
- Silence (DAC output only)

These sources can be routed to play out the DAC or USB record output.

4 Understanding microphone configurations

The critical components in Qualcomm ANC enabled Headsets or Earbuds are the selection of microphones, the associated Qualcomm ANC mode, and the microphone usage for other applications.

4.1 ANC microphone configuration guidelines

A set of guidelines for microphone and ANC configurations, are listed in [Table 4-1](#).

Table 4-1 Rules for microphone and ANC configurations

Rule	Guidelines
General	▪ Feed-forward throughout or feedback ANC requires one microphone per earpiece. ▪ Qualcomm hybrid ANC requires two microphones per earpiece.
Analog mics with Qualcomm ANC	▪ ANC microphones can be analog. ▪ One or both analog microphone inputs on the chip can be used. ▪ Audio instance 0 or instance 1 can be used for QCC517x. ▪ The analog microphone gain is shared with other DSP processing. ▪ The ANC microphones are agnostic to the sample rate of the codec (8 kHz/16 kHz)
Digital mics with Qualcomm ANC	▪ ANC microphones can be digital. ▪ Digital microphones can be on instance 0,1,2. ▪ Instance 0,1,2,3 can be used for QCC517x. ▪ All digital microphone channels can support ANC. ▪ No digital microphone codec gain is provided for ANC. The ANC block provides its own digital gain. ▪ The ANC microphone is agnostic to the sample rate of the codec (8 kHz/16 kHz). ▪ ANC supports a DMIC clock rate of 2 MHz and 4 MHz. ▪ ANC microphones can be the same audio instance or different audio instance.
Mic sharing with cVc uplink processing	▪ One or multiple microphones can be shared with with cVc uplink processing. ▪ Both the internal or external microphone can be if the DSP processing supports it. ▪ All microphones that provide an input for cVc processing must be of the same type (analog/digital) and the same model/part number. ▪ The use of digital cVc can be across the same audio instance, or different audio instance.

Table 4-1 Rules for microphone and ANC configurations (cont.)

Rule	Guidelines
Mixing microphone types with Qualcomm ANC	A mixture of analog and digital mics is supported.
Qualcomm ANC tuning mode	- Not required in mission mode (Qualcomm ANC ON/OFF in product). - One of microphone instance 0 or 1 must be used for data capture from inside the Qualcomm ANC block.

4.2 Stereo headset Qualcomm hybrid ANC use case (Static ANC)

There are many microphone combinations across the applications. In [Table 4-2](#), each column represents a stereo headset Qualcomm hybrid ANC use case.

Table 4-2 Stereo headset hybrid ANC use case

Mic instance		Stereo headset	Stereo headset	Stereo headset	Stereo headset
-	ANC	Hybrid	Hybrid	Hybrid	Hybrid
Instance 0	ANA_MIC1	–			ANC FB
	ANA_MIC2	–	–		ANC FB
Instance 0	DIG_MIC1	ANC FF	–		–
	DIG_MIC2	ANC FB	–		–
Instance 1	DIG_MIC3	ANC FF	ANC FF	ANC FF	ANC FF
	DIG_MIC4	ANC FB	ANC FF	ANC FB	ANC FF
Instance 2	DIG_MIC5	–	ANC FB	ANC FF	–
	DIG_MIC6	–	ANC FB	ANC FB	–
Description	–	All mics on digital mic instance	All mics on digital mic instance	All mics on digital mic instance	FF ANC mics on digital mic instance and FB mics on analog mic instance

NOTE From QCC517x/QCC307x devices onwards, support Instance 1 is for analog microphone, the resulting permutation and combination is not shown for the same.

4.3 Mono earbud Qualcomm hybrid ANC use case (Static, Adaptive ANC, and Enhanced Adaptive ANC)

In [Table 4-3](#), each column represents a mono earbud Qualcomm hybrid ANC use case. Use this as a reference for earbud use cases that can be built with the Earbud Application. This information is relevant for both Static and Adaptive ANC.

Table 4-3 Mono Earbud Qualcomm hybrid ANC use case

Mic instance		Earbud	Earbud	Earbud	Earbud	Earbud
-	ANC	Hybrid	Hybrid	Hybrid	Hybrid	Hybrid
Instance 0	ANA_MIC1	–	–	–	ANC FB	ANC FF
	ANA_MIC2	–	–	–	–	ANC FB
Instance 0	DIG_MIC1	ANC FF	–	ANC FF	–	–
	DIG_MIC2	ANC FB	–	–	–	–
Instance 01	DIG_MIC3	–	ANC FF	ANC FB	ANC FF	–
	DIG_MIC4	–	ANC FB	–	–	–
Instance 2	DIG_MIC5	–	–	–	–	–
	DIG_MIC6	–	–	–	–	–
Description	–	All mics on digital mic instance	ANC mics on digital mic instance	All mics on digital mic instance	FF ANC mic on digital mic instance and FB mic on analog mic instance	ANC mics on analog instance

NOTE Instance 1 is for analog microphone, the resulting permutation and combination is not shown.

NOTE For Enhanced ANC, ADK 20.3 supports only hybrid mode configuration for QCC514x and QCC515x devices. ADK 21.1 additionally supports enhanced ANC software in feed-forward configuration for QCC515x variant.

5 ANC development process

Figure 5-1 shows the steps involved in configuring ANC and the tools used at each stage of the process.

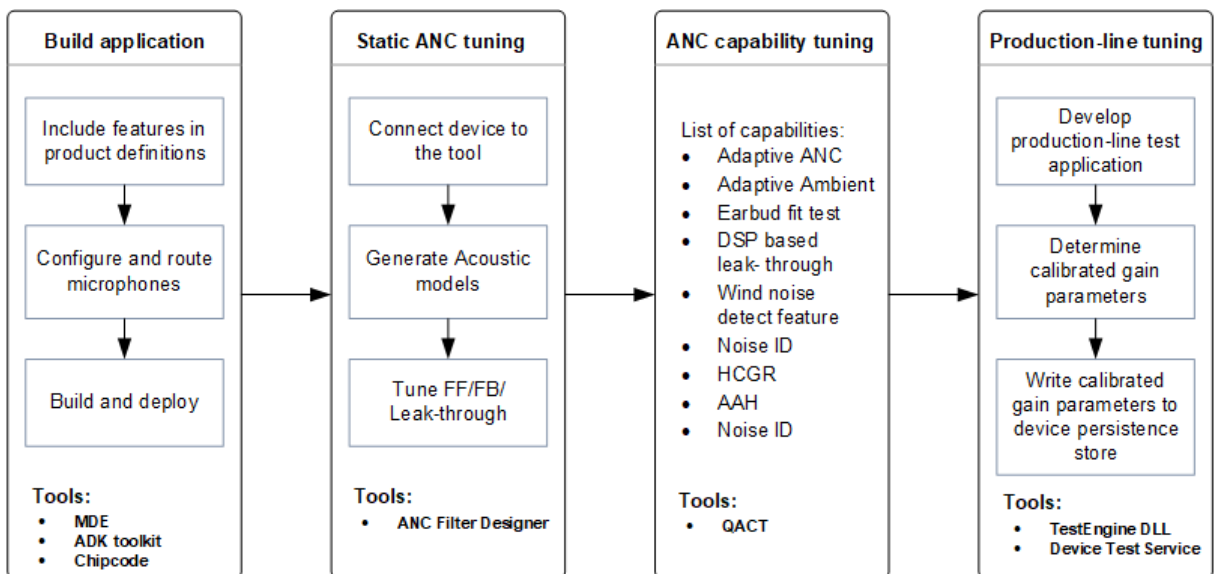


Figure 5-1 ANC development process and tools

Build application

The Earbud Application supports ANC feed-forward, feedback and Hybrid modes. To set up ANC, enable the ANC run mode in the Earbuds Application. Add the required ANC features to the Earbuds project and then build the application, as described in [Configure ANC in the Earbuds Application](#).

The ANC configuration can support either digital/analog microphones or mix of analog and digital microphones, which can be configured by using various settings, as described in [Route hardware mic instances to software mic instances](#).

Static ANC tuning

Static ANC tuning involves designing the ANC feed-forward, feedback, EC, and leak-through IIR filters, which are static in nature. They are not updated or changed once they are saved to ANC file system.

NOTE In Adaptive ANC mode, the adaptive algorithm only updates the gain. The other filter parameters from static ANC stay the same as saved in the ANC file system.

For a given headset, primary path (passive attenuation), secondary path (headset speaker response) acoustic models are generated to arrive at a target ANC filter model. The ANC filter designer tool uses this target model to generate ANC IIR filters.

For more information, see [Static ANC tuning using ANC Filter Designer](#).

ANC capability tuning

The Qualcomm ADK includes various audio capabilities or functionalities, for example Earbud Fit Test and DSP based leak-through that are supported on the chip. Configure the required capabilities, as described in [Adaptive ANC tuning use cases](#).

Production-line tuning or calibration

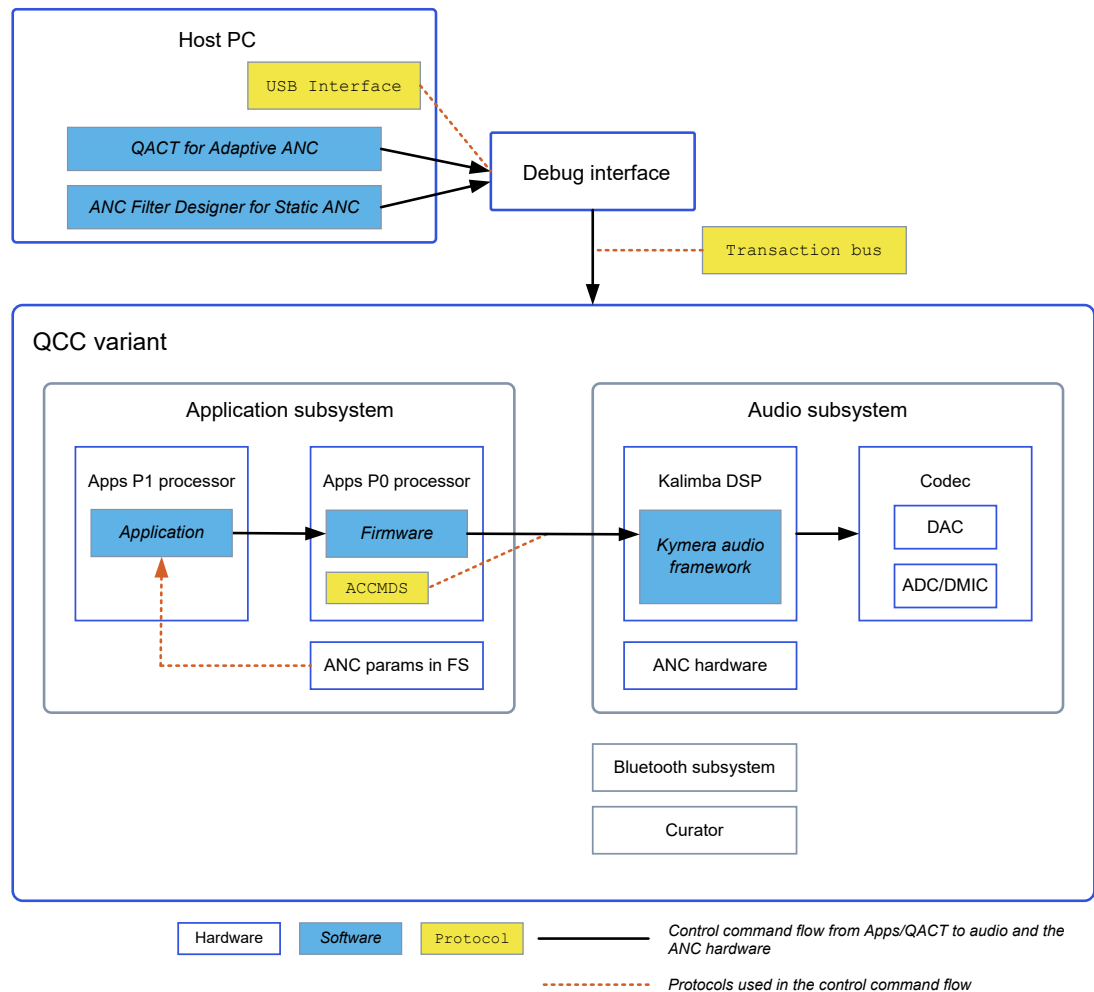
Production-line tuning involves calibration of ANC fine gain on a sealed earbud over UART transport. The feature provides a reference code for after plastic calibration of ANC gain parameters, which needs to be used in sync with customer-defined audio measurement setup and tools. For production line calibration with a sealed headset, ANC gains are fine-tuned based on variations in plastics and transducer tolerances of the microphone and speaker.

For more information, see [Production-line calibration](#).

5.1 ANC tuning setup

The ANC tuning setup includes the QCC variant, hardware interfaces, software tools, and protocols that enable communication between the Earbud Application and the ANC hardware. Figure 5-2 shows the high-level ANC interface block diagram and the interaction between various components in a typical ANC setup.

Figure 5-2 High-level ANC interface block diagram



QCC variant

The QCC517x, QCC307x, QCC518x, and QCC308x variants have multiple subsystems on the chip, described in [Table 5-1](#).

Table 5-1 Subsystems on a QCC variant

Subsystem	Description
Application subsystem	The Earbud Application runs on the Application subsystem. The application can set up and control audio processing functions running on the Audio subsystem.
Audio subsystem	The Audio subsystem runs Kymera system DSP framework, which is a platform-independent software framework and provides a modular approach to linking audio processing blocks, also known as capabilities. Kymera enables customer applications to achieve a variety of audio processing use cases.
Bluetooth subsystem	Dual-mode radio, which supports concurrent classic and Bluetooth low energy operation. It is fully qualified to the Bluetooth v5.2 specification.
Audio CPU commands (ACCMDS)	Provides a mechanism for the host or chip subsystems to communicate with and command the Audio subsystem on the QCC517x or QCC307x chip.
Transaction bus	Connects the debug interface to the QCC variant.

In addition, the chip also runs various software components, which are described in [Table 5-2](#).

Table 5-2 Software components running on a QCC variant

Subsystem	Description
Earbud Application	Reference application software, designed to work end-to-end with reference smartphone applications.
Firmware	Used for communication using trap APIs (sending and receiving messages over TBUS).
Kymera audio framework	Platform-independent software framework that is designed to facilitate the usage of audio capabilities and control the underlying hardware and memory management.

The hardware setup also includes the physical debug interface, which is a host interface that provides connection to a host PC. The debug interface is connected to the QCC variant using a transaction bus.

Host PC and software

The Earbuds Application can be configured by using the Qualcomm MDE, which runs on the host PC.

Additionally, the host PC runs the software tools, which are required for ANC tuning and calibration:

- *Qualcomm Audio Calibration Tool (QACT)*, used for adaptive ANC tuning, which involves tuning capabilities such as Qualcomm Adaptive ANC, HGCR, and Earbud fit test.
- *ANC Filter Designer Tool*, used for Static ANC tuning.

Debug interface

The Debug interface allows communication between the host PC and the QCC variant. The interface supports the following functions:

- Deployment of the application to the hardware
- Debugging the application

- Logging application events
- Tuning configuration data
- Profiling audio processing
- Retrieving core dump logs
- Factory programming and production test

6 Configure ANC in the Earbuds Application

The Earbud Application supports ANC feed-forward, feedback and hybrid modes. The ANC configuration can support digital microphones, analog microphones, or mix of analog and digital microphones.

Figure 6-1 shows the configuration tasks involved in building an Earbuds Application with ANC.

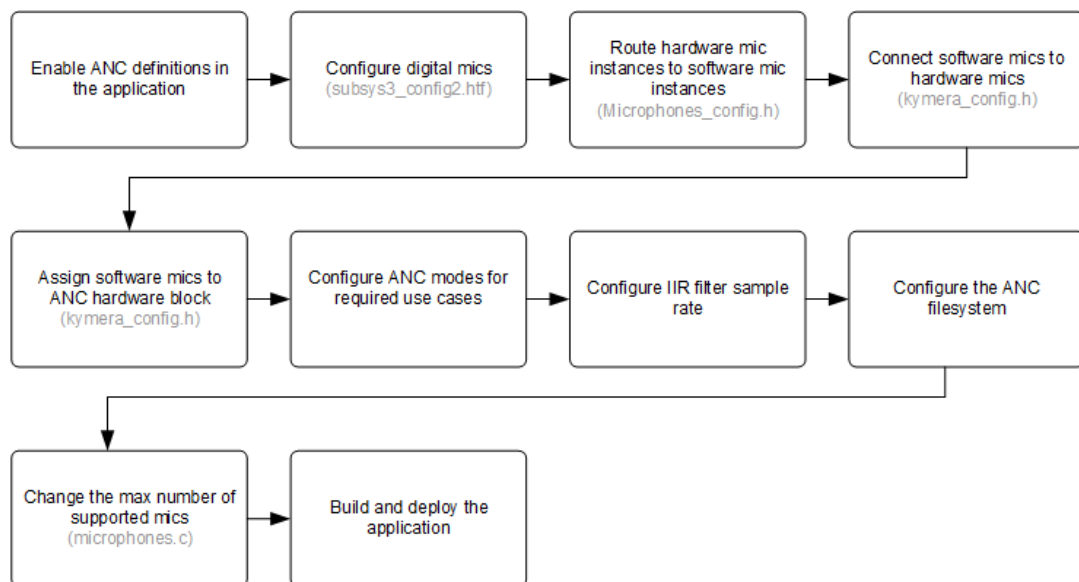


Figure 6-1 ANC configuration process

6.1 Enable ANC features in the Earbuds application

To enable ANC features in the Earbuds Application, add the required ANC definitions to the Earbuds project properties in Qualcomm MDE.

1. Launch Qualcomm MDE. Go to **MDE projects > General > DEFS**.
2. In the Earbuds project properties, add the required ANC definitions listed in [Table 6-1](#).

Table 6-1 Definitions for enabling Qualcomm ANC, Enhanced ANC, Adaptive ANC, and Enhanced Adaptive ANC

Feature	Definitions to add in the Earbuds project properties
Static ANC	ENABLE_ANC
Static Enhanced ANC	ENABLE_ANC ENABLE_ENHANCED_ANC
Adaptive ANC	ENABLE_ANC ENABLE_ADAPTIVE_ANC pre-ADK 21.1: Also add INCLUDE_MIC_CONCURRENCY
Enhanced Adaptive ANC	ENABLE_ANC ENABLE_ENHANCED_ANC ENABLE_ADAPTIVE_ANC
Enhanced Adaptive Dual ANC	ENABLE_ANC ENABLE_ENHANCED_ANC ENABLE_ADAPTIVE_ANC ENABLE_ANC_DUAL_FILTER_TOPOLOGY
ANC tuning mode	With definitions added for each use case above, add the following additional definitions: ENABLE_ANC_TUNING_MODE ENABLE_USB_AUDIO

3. *Qualcomm Adaptive ANC or Enhanced Adaptive ANC only:* Add the AANC capability download_aanc.edcks file into the ro_fs project.
 - Right-click **ro_fs**. Select **Add Existing Files...**, and then click **goto**.
 - Select the <ADK>\audio\qcc514x_qcc304x\kalimba_ROM_7120\kymera\prebuilt_dkcs\streplus_rom_release\download_aanc.edcks file.
4. Build the application. Go to **Build**, and then click **Deploy All**.

6.2 Configure digital microphones

To configure digital microphones in the Earbuds Application, define the microphone parameters in the `subsys3_config2.htf` file.

Table 6-2 lists the parameters that can be configured.

Table 6-2 Digital microphone parameters

MIB key	Value	Description
DigMicXG722FilterEnable	True	Enables optional G722 filter in digital microphones to improve noise performance.
DigMicXChanSwap	False	If set, the two channels of digital microphones of <code>audio_instance_X</code> are swapped. Each available audio instance ($X = 0, 1, 2, \dots$) can be configured to various parameters. For information about audio instances, see the <i>Audio Interfaces</i> section of the <i>Datasheet</i> .
DigMicXClockRate	Clock rate	Digital mic clock rate, for example, 4000 for 4 MHz clock rate. Currently 2 MHz and 4 MHz are supported. The DMIC clock and data pins can be assigned to any available PIO. For more information about the available PIOs, see the <i>Datasheet</i> and <i>Schematic</i> .
DigMicXPioConfig	[C D]	Pin C is the <code>CLK_OUT</code> and pin D is the <code>DATA_IN</code> . If the audio instance is not connected to digital mics, it should be set to <code>0xFF</code> .
AncIIRFilterSampleRate	1600 0, 3200 0, 6400 0	ANC IIR filter sample rate in Hz. To design filter that can reduce out of band noises, Qualcomm recommend using a value of 32000 or 64000 .
Codec0OutputQuietModeControl	1,2,3	DAC gain: 1: 0 dB gain 2: -6 dB gain 3: -12 dB gain This value takes effect for class AB audio output only to reduce idle noise. The DAC gain can also be configured using a MIB key.

Example: Configuring digital mics in audio instance 2

The following example shows settings for configuring digital mics in audio instance 2 `DigMic2_`, where `PIO-18 (0x12)` is the `CLK_OUT`, `PIO19 (0x13)` is the `DATA_IN`, and the sampling rate is 4 MHz.

```
DigMic2G722FilterEnable = true
DigMic2G722FirEnable = true
DigMic2ChanSwap = false
DigMic2ClockRate = 4000
```

```
DigMic2PioConfig = [12 13]
AncIIRFilterSampleRate=32000
```

6.3 Route hardware mic instances to software mic instances

To route the hardware microphone instances to the software ANC microphone instances, edit the `microphones_config.h` file in the `<ADK path> \src\domains\audio\microphones` location, and then select the required settings, listed in [Table 6-3](#).

Table 6-3 ANC microphone settings

Setting	Value	Description
Mic Bias	■ <code>BIAS_CONFIG_DISABLE</code>: Disable bias config. ■ <code>BIAS_CONFIG_MIC_BIAS_0</code>: Analog Bias 0 from the chip. ■ <code>BIAS_CONFIG_MIC_BIAS_1</code>: Analog Bias 1 from the chip ■ <code>BIAS_CONFIG_PIO</code>: Set bias driven to a PIO (DMIC or AMIC can be used)	The bias type driven MicX
Mic Bias Voltage	■ For analog mic, setting a value from following: 0 (2.1 V), 1 (2.1 V), 2 (2.0 V), 3 (1.9 V), 4 (1.8 V), 5 (1.7 V), 6 (1.6 V), 7 (1.5 V) ■ For digital mic, this bias voltage is ignored.	Sets the bias voltage for MicX.
Mic Bias PIO Configuration	Any available PIO.	Used only if <code>BIAS_CONFIG_PIO</code> is configured as source of Mic Bias.
Mic Gain	—	—
Microphone Type	■ <code>mic_type_analog</code> ■ <code>mic_type_digital</code>	Sets the mic type for the MicX, either analog or digital.
Mic Audio Instance	■ <code>AUDIO_INSTANCE_0</code>: Supports both analog and digital microphones. ■ <code>AUDIO_INSTANCE_1</code>: □ Supports only digital microphones. □ QCC517x and QCC307x devices support analog microphone and provide the ability to run four analog microphones simultaneously. ■ <code>AUDIO_INSTANCE_2</code>: Supports only digital microphones. ■ <code>AUDIO_INSTANCE_3</code>: □ Supports only digital microphones. □ Applicable for only QCC517x and QCC307x devices.	Routes MicX to an audio instance.
Mic Audio Channel	■ <code>AUDIO_CHANNEL_A</code> ■ <code>AUDIO_CHANNEL_B</code>	Route MicX to a channel from the chosen audio instance.

Analog microphone example

```
#define appConfigMic0Bias()          (BIAS_CONFIG_MIC_BIAS_0)
#define appConfigMic0BiasVoltage()  (3)
#define appConfigMic0Pio()          (0x0)
#define appConfigMic0Gain()         (0x5)
#define appConfigMic0IsDigital()    (FALSE)
#define appConfigMic0AudioInstance() (AUDIO_INSTANCE_0)
#define appConfigMic0AudioChannel() (AUDIO_CHANNEL_A)

#define appConfigMic1Bias()          (BIAS_CONFIG_MIC_BIAS_0)
#define appConfigMic1BiasVoltage()  (3)
#define appConfigMic1Pio()          (0x0)
#define appConfigMic1Gain()         (0x5)
#define appConfigMic1IsDigital()    (FALSE)
#define appConfigMic1AudioInstance() (AUDIO_INSTANCE_0)
#define appConfigMic1AudioChannel() (AUDIO_CHANNEL_B)
```

Digital microphone example

```
#define appConfigMic2Bias()          (BIAS_CONFIG_PIO)
#define appConfigMic2BiasVoltage()  (3)
#define appConfigMic2Pio()          (19)
#define appConfigMic2Gain()         (15)
#define appConfigMic2IsDigital()    (TRUE)
#define appConfigMic2AudioInstance() (AUDIO_INSTANCE_1)
#define appConfigMic2AudioChannel() (AUDIO_CHANNEL_A)

#define appConfigMic3Bias()          (BIAS_CONFIG_PIO)
#define appConfigMic3BiasVoltage()  (3) /* 1.9v */
#define appConfigMic3Pio()          (19)
#define appConfigMic3Gain()         (15)
#define appConfigMic3IsDigital()    (TRUE)
#define appConfigMic3AudioInstance() (AUDIO_INSTANCE_1)
#define appConfigMic3AudioChannel() (AUDIO_CHANNEL_B)
```

6.4 Connect software mics to ANC mics

To connect the software microphones to the ANC microphones for Qualcomm ANC, enhanced ANC, or cVc, edit the <ADK path>\src \domains\audio\kymera\kymera_config.h file.

Table 6-4 lists the settings to use.

Table 6-4 Settings for connecting microphones

Function	Description	Values
#define appConfigAncPathEnable()	Defines the ANC mode.	■ hybrid_mode_left_only ■ feed_forward_mode_left_only ■ feed_back_mode_left_only ■ hybrid_mode

Table 6-4 Settings for connecting microphones (cont.)

Function	Description	Values
		■ feed_forward_mode ■ feed_back_mode
#define appConfigAncFeedForwardMic()	Defines the feed-forward mic	microphones_X
#define appConfigAncFeedBackMic()	Defines the feedback mic.	microphones_X
#define appConfigAncTuningMonitorMic()	Defines the tuning mic if the tuning mode is built.	microphones_X Ensure that the ANC mics(FF/FB mics) and tuning mic are assigned in different audio instances. Assign the tuning mic in instance 0 or instance 1. The tuning mic audio instance is used for collecting recordings within the ANC hardware and does not require physical mic to be connected.
#define appConfigAncTuningEnabled()	Enables the ANC tuning mode.	TRUE/FALSE

Hybrid mode example

The following example shows settings that define the ANC mode as hybrid mode.

```
#define appConfigAncPathEnable()                (hybrid_mode_left_only)
#define appConfigAncFeedForwardMic()            (microphone_5) → FF mic:
microphone_5 is Mic4 in microphones_config
#define appConfigAncFeedBackMic()              (microphone_2) → FB mic:
microphone_2 is Mic1 in microphones_config.h
```

Tuning mode example

If ANC tuning mode is built, use the following configuration:

```
#define appConfigAncTuningMonitorMic()          (microphone_3) → Monitor mic:
microphone_3 is Mic2 in microphones_config.h
#define appConfigAncTuningEnabled()             (TRUE) → Enable ANC tuning mode
```

NOTE If tuning is performed in mission mode, there is no requirement to build the ANC tuning mode.

6.5 Assign microphones for Qualcomm ANC or enhanced ANC (Run mode)

To assign microphones for Qualcomm ANC, enhanced ANC, or cVc, edit the <ADK path>\src\domains\audio\kymera\kymera_config.h file.

The following code shows an example for assigning microphones in a Hybrid ANC with 1-mic cVc sharing and mixed mic configuration use case.

```

/*!@{ @name ANC configuration for hybrid mode */
#define appConfigAncPathEnable ()      (hybrid_mode_left_only)
/*! Disable ANC tuning functionality */
#define appConfigAncTuningEnabled()    (FALSE)
#define appConfigAncTuningMonitorMic() (microphone_none)

/* Use microphone 3 as ANC feed-forward mic */
#define appConfigAncFeedForwardMic()  (microphone_3)
/* Use microphone 1 as ANC feed-back mic */
#define appConfigAncFeedBackMic()     (microphone_1)

/* ANC filter mode selection in default case */
#define appConfigAncMode()             (anc_mode_1) /*!@}

/*! microphone to use for the first SCO mic */
#define appConfigScoMic1() (microphone_3)
/*! microphone to use for the second and third SCO mics. These should be
defined as microphone_none if using 1-mic CVC */
#define appConfigMicExternal() (microphone_none)
#define appConfigMicInternal() (microphone_none)

```

6.6 Assign microphones for Qualcomm ANC or enhanced ANC (Tuning mode)

ANC tuning mode requires two microphone audio instances. One microphone audio instance is used for feed-forward and feedback microphones, and another instance is used for the tuning monitor microphone.

ANC microphones and tuning monitor microphones should be assigned in different microphone audio instances. The tuning monitor microphone audio instance is used for collecting recordings within the ANC hardware and does not require the physical microphone hardware to be connected.

Microphone audio instances to use

Configure the ANC tuning monitor microphone, as described in [Table 6-5](#).

Table 6-5 ANC microphone audio instances

Microphone type	Mic instance
Digital	0
Analog	1

NOTE Parallel filter configuration (enhanced ANC) does not support different path recording using the ANC tuning capability.

Hybrid ANC example (configured using digital microphones)

```

/*!@{ @name ANC configuration for hybrid mode */
#define appConfigAncPathEnable()          (hybrid_mode_left_only)

/* Enable ANC tuning functionality */
#define appConfigAncTuningEnabled()        (TRUE)
/* Use microphone 1 for monitor microphone */
#define appConfigAncTuningMonitorMic()     (microphone_1)

/* Use microphone 3 (digital mic) as ANC feed-forward mic */
#define appConfigAncFeedForwardMic()       (microphone_3)
/* Use microphone 4 (digital mic) as ANC feed-back mic */
#define appConfigAncFeedBackMic()          (microphone_4)

```

Tuning

To tune ANC:

1. Connect the USB charger to the earbud device. The earbud device tries to enumerate as a speaker and microphone device in host PC. Any active audio connections in earbud device are dropped and an ANC tuning chain is created.
2. Use QACT to tune ANC.
3. After ANC tuning is complete, reset the device for the ANC tuning parameters to be applied.

6.7 Configure ANC modes for the required use cases

The Earbud Application supports various ANC use cases including Static Noise cancellation, Static Ambient/leak-through, Adaptive Noise cancellation, and Adaptive Ambient/leak-through (see [Adaptive ANC tuning use cases](#)). ANC modes 1 to 10 can be used for different filter bank settings across Static and Adaptive versions of ANC. Each mode can be configured to follow particular ANC role.

For configuration purposes, ANC includes a static two-dimensional array.

NOTE Update this table in synchronization with the ANC tuning defined for the product. Qualcomm also recommend configuring the modes in sequence.

To configure modes:

1. Update the `appConfigNumOfAncModes()` (`kymera_config.h`) file to the configured number of modes. The tuning files are `anc_tuning_config.htf` and `aanc_parameters.htf`.
2. Associate a role with an ANC mode by modifying the two-dimensional array variable `anc_config_data` in the `.../adk/src/domains/audio/anc/anc_config.c` file.

In the default settings, ANC mode 1 is assigned to Adaptive Noise cancellation. The remaining ANC modes are associated with the Static Noise cancellation role.

To define role modes such as: Mode 1- Adaptive noise cancellation, Mode 2- Adaptive Ambient, Mode 3- Static Ambient/Leakthrough, Mode 4 to Mode 10- Static Noise Cancellation, modify the default settings, as shown in the following example.

```
const anc_config_data_t anc_config_data[ANC_CONFIG_MAX_MODE] =
{
    {ANC_CONFIG_MODE_ADAPTIVE, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_1*/
    {ANC_CONFIG_MODE_ADAPTIVE, ANC_CONFIG_MODE_LEAKTHROUGH}, /*anc_mode_2*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_LEAKTHROUGH }, /*anc_mode_3*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_4*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_5*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_6*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_7*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_8*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_9*/
    {ANC_CONFIG_MODE_STATIC, ANC_CONFIG_MODE_NOISE_CANCELLATION}, /*
*anc_mode_10*/
};
```

6.8 Configure IIR filter sample rate

The IIR filter can be run at 32 kHz or 64 kHz sampling rate. Qualcomm recommend that the filters run at 32 kHz . Depending on the filter tuning, set the ANC IIR filter sample rate in the `subsys3_config2.htf` file in the `fw_cfg` location.

For example:

```
AncIIRFilterSampleRate = 64000
```

Ensure that this value is consistent with the filter sample rate configured in the application. For example, in the file `.../adk/src/domains/audio/kymera_anc_advanced/kymera_adaptive_anc.c`, define `AANC_FILTER_SAMPLE_RATE` (`adaptive_anc_sample_rate_64khz`).

6.9 Configure the filesystem for ANC or enhanced ANC

To configure the filesystem for ANC or enhanced ANC, check that the keys found within *.htf files in ADK installer path `_filesystem(fw_cfg -> subsys3_config2.htf)` project are configured and accurate for the particular hardware variant used.

Complete the following tasks:

1. Optionally, set up **AncEnableDelay** for click/pop free ANC start up behavior.
2. Comment or uncomment relevant keys in **user_ps_filesystem > anc_tuning_config.htf** based on the required ANC mode – feed-forward, feedback, or hybrid.
3. Go to **Build**, and then click **Deploy All**. Load the same image on both earbuds except `bdaddr`; only the bluetooth address remains different on the individual earbuds.

6.10 Configure filesystem for Qualcomm Adaptive ANC or enhanced Adaptive ANC

To configure the file system for Adaptive ANC or enhanced Adaptive ANC, check that the ANC file system configuration defined for the static ANC configuration is available and configured and accurate for the particular hardware variant used.

Complete the following tasks:

1. Ensure that the tuned parameters for Adaptive ANC, in the `ps_cfg > aanc_parameters.htf` file, are available.
2. Go to **Build**, and then click **Deploy All**. Load the same image on both earbuds except `bdaddr`; only the bluetooth address remains different on the individual earbuds.

6.11 Enable dynamic ANC filter transition

ANC mode switch should happen without turning off the the ANC while avoiding audio artifacts in some usecases. This dynamic ANC filter transition has the ability to switch ANC modes smoothly and quickly using delta ramping algorithm (first order low pass filter) for both ramp up and ramp delay.

To enable this functionality, add the flag `ENABLE_ANC_FAST_MODE_SWITCH` to **Project Defs** in the MDE Earbud project.

With this mode enabled:

1. When the ANC modes have completely different filter coefficients (both FF and FB), a gain ramp down and ramp up process is necessary for loading the new coefficients and avoid the audio artifacts. There is also an improvement in mode switch time.
2. When the ANC modes have same filter coefficients at least in one ANC path(FF, FB or both), the AHM capability will keep the coefficients untouched and ramp the previous gain to new gain directly. In this case, it will always keep some ANC performance in the background during the switch process. Most of the use cases follows this, as they keep the same FF filter but use different FB filter for different ANC modes.

The tuning of the mode change time can be performed with the parameters of AHM capability using QACT, shown in the figure. Select the parameters based on the comfort of end user while avoiding the audio artifacts.

Fast ANC Mode Change: Ramp Down Duration	0x00006666	0.02499 s
Fast ANC Mode Change: Ramp Up Duration	0x00006666	0.02499 s
Fast ANC Mode Change: Delay Duration	0x000028F6	0.01000 s
Slow ANC Mode Change: Ramp Down Duration	0x00040000	0.25000 s
Slow ANC Mode Change: Ramp Up Duration	0x00040000	0.25000 s
Slow ANC Mode Change: Delay Duration	0x000028F6	0.01000 s

Figure 6-2 AHM capability parameters

Table 6-6 AHM capability parameters

Parameter name	Description
Fast mode ANC change	ANC mode switch method to switch between similar ANC types. For example, noise cancelling mode to another noise cancelling mode or leak-through mode to another leak-through mode.
Slow mode ANC change	ANC mode switch method to switch between different ANC types. For example, noise cancelling mode to leak-through mode or vice-versa.
Ramp down duration	Duration of the fine gain ramp into mute.
Ramp up duration	Duration of the fine gain ramp into static tuned value.
Delay duration	Delay before the fine gain ramp.

6.12 Build and deploy the application

To build and deploy the Earbud Application, launch Qualcomm MDE. Go to **Tools > Build**, and then click **Deploy All Projects**.

7 Perform audio curation and debugging

To perform audio curation and debugging, use either the Pydbg commands or the GAIA application.

7.1 Debugging with Pydbg

Table 7-1 shows the Pydbg test functions for debugging the ANC functionality in the Earbuds Application.

Table 7-1 Debugging with Pydbg

Pydbg Test functions	Functionality
<code>apps1.fw.call.appTestPhyStateOutOfCaseEvent()</code>	Place EB into out of case
<code>apps1.fw.call.appTestPhyStateInCaseEvent()</code>	Place EB into In case
<code>apps1.fw.call.EarbudTest_SetAncEnable()</code>	Switch on Static ANC on both earbuds
<code>apps1.fw.call.EarbudTest_SetAncDisable()</code>	Switch off Static ANC on both earbuds
<code>apps1.fw.call.EarbudTest_SetAncToggleOnOff()</code>	Toggle Static ANC ON/OFF on both earbuds
<code>apps1.fw.call.EarbudTest_SetAncMode(mode)</code>	Set filter bank modes on both earbuds between 0 (anc_mode_1) to 9 (anc_mode_10)
<code>apps1.fw.call.EarbudTest_SetNextAncMode()</code>	Set next ANC mode on both earbuds
<code>apps1.fw.call.EarbudTest_GetAncstate()</code>	Get current ANC State of the earbud
<code>apps1.fw.call.EarbudTest_GetAncMode()</code>	Get current ANC mode of the earbud
<code>appTestGetAdaptiveAncFeedForwardGain()</code>	Get current adaptive ANC converged feed-forward mic path filter gain
<code>appTestSetAdaptiveAncGainParameters()</code>	Set adaptive ANC gain parameters in feed-forward mic path filter gain
<code>apps1.fw.call.EarbudTest_ToggleAncConfig()</code>	Test the toggle behaviour configured through GAIA app or through Pydbg commands
<code>apps1.fw.call.EarbudTest_GetAncToggleConfig1()</code>	Get the current value of Toggle Config 1
<code>apps1.fw.call.EarbudTest_SetAncToggleConfig1()</code>	Set the value of Toggle Config 1
<code>apps1.fw.call.EarbudTest_GetAncToggleConfig2()</code>	Get the current value of Toggle Config 2
<code>apps1.fw.call.EarbudTest_SetAncToggleConfig2()</code>	Set the value of Toggle Config 2

Table 7-1 Debugging with Pydbg (cont.)

Pydbg Test functions	Functionality
<code>apps1.fw.call.EarbudTest_GetAncToggleConfig3()</code>	Get the current value of Toggle Config 3
<code>apps1.fw.call.EarbudTest_SetAncToggleConfig3()</code>	Set the value of Toggle Config 3
<code>apps1.fw.call.EarbudTest_GetAncConfigDuringStandalone()</code>	Get the current value of config to be used during Standalone ANC
<code>apps1.fw.call.EarbudTest_SetAncConfigDuringStandalone()</code>	Set the config to be used during Standalone
<code>apps1.fw.call.EarbudTest_GetAncConfigDuringPlayback()</code>	Get the current value of config to be used during Playback/Music
<code>apps1.fw.call.EarbudTest_SetAncConfigDuringPlayback()</code>	Set the config to be used during Playback/Music
<code>apps1.fw.call.EarbudTest_GetAncConfigDuringSco()</code>	Get the current value of config to be used during SCO/HFP call
<code>apps1.fw.call.EarbudTest_SetAncConfigDuringSco()</code>	Set the config to be used during SCO/HFP call
<code>apps1.fw.call.EarbudTest_GetAncConfigDuringVoiceAssistant()</code>	Get the current value of config to be used during Voice Assistant
<code>apps1.fw.call.EarbudTest_SetAncConfigDuringVoiceAssistant()</code>	Set the config to be used during Voice Assistant
<code>apps1.fw.call.EarbudTest_GetAncDemoState()</code>	Get if Demo state is active
<code>apps1.fw.call.EarbudTest_SetAncDemoState()</code>	Set Demo state to active
<code>apps1.fw.call.appTestGetCurrentAdaptiveAncV2Mode()</code>	QCC517x/QCC307x devices: Get the ANC system mode
<code>apps1.fw.call.EarbudTest_GetCurrentAhmMode()</code>	QCC517x/QCC307x devices: Get the ANC hardware manager Mode
<code>apps1.fw.call.appTestGetAhmFFGain()</code>	QCC517x/QCC307x devices: Get the ANC hardware manager gain
<code>apps1.fw.call.EarbudTest_SetWorldVolumeUp()</code>	QCC517x/QCC307x devices: Increases world volume when ANC mode is configured with Adaptive Ambient role
<code>Apps1.fw.call.EarbudTest_SetWorldVolumeDown()</code>	QCC517x/QCC307x devices: Decreases world volume when ANC mode is configured with Adaptive Ambient role

7.2 GAIA application for audio curation

From ADK 21.x onwards, the ANC feature Id (0x02) in GAIA has been deprecated and a new feature ID (0x08) is used for audio curation.

From ADK 21.x onwards, the GAIA application provides the following features:

- Option to configure the toggle buttons to the ANC modes. Currently, the maximum toggle configuration supported is three. To move through the configured modes of ANC, use the toggle button on the device.
- Options to configure the behavior of ANC during concurrency scenarios such as during a call, while streaming music or using Digital Assistant.
 - During concurrency mode, for example during an ANC during a phone call use case, or ANC with music use case, the ANC can be switched to a mode that can be configured by using the GAIA application.
 - The option **same as toggle config** does not change the behavior of ANC during the concurrency scenarios.
 - The toggle button can be used during the concurrency scenarios.

The file `kymera_config.h` defines the default values for the product. These can be changed or configured through GAIA app. The configured values persist in the device in `PS_KEY_ANC_SESSION_DATA`.

```

/! Configure Toggle behavior.*/
#define ancConfigToggleWay1()      (anc_toggle_config_mode_1)
#define ancConfigToggleWay2()      (anc_toggle_config_mode_5)
#define ancConfigToggleWay3()      (anc_toggle_config_not_configured)

/! Configure ANC modes to be used for concurrency cases.*/
#define ancConfigStandalone()
(anc_toggle_config_is_same_as_current)
#define ancConfigPlayback()
(anc_toggle_config_is_same_as_current)
#define ancConfigVoiceAssistant()
(anc_toggle_config_is_same_as_current)
#define ancConfigVoiceCall()
(anc_toggle_config_is_same_as_current)

```

8 Adaptive ANC and Adaptive Ambient tuning using Qualcomm Audio Calibration Tool

To configure ANC, connect the device to the Qualcomm Audio Calibration Tool (QACT). QACT is a standalone PC application that calibrates audio processing capabilities within the Kymera audio framework running on the device. QACT is used for tuning Adaptive ANC and Adaptive Ambient ANC.

QACT user interface

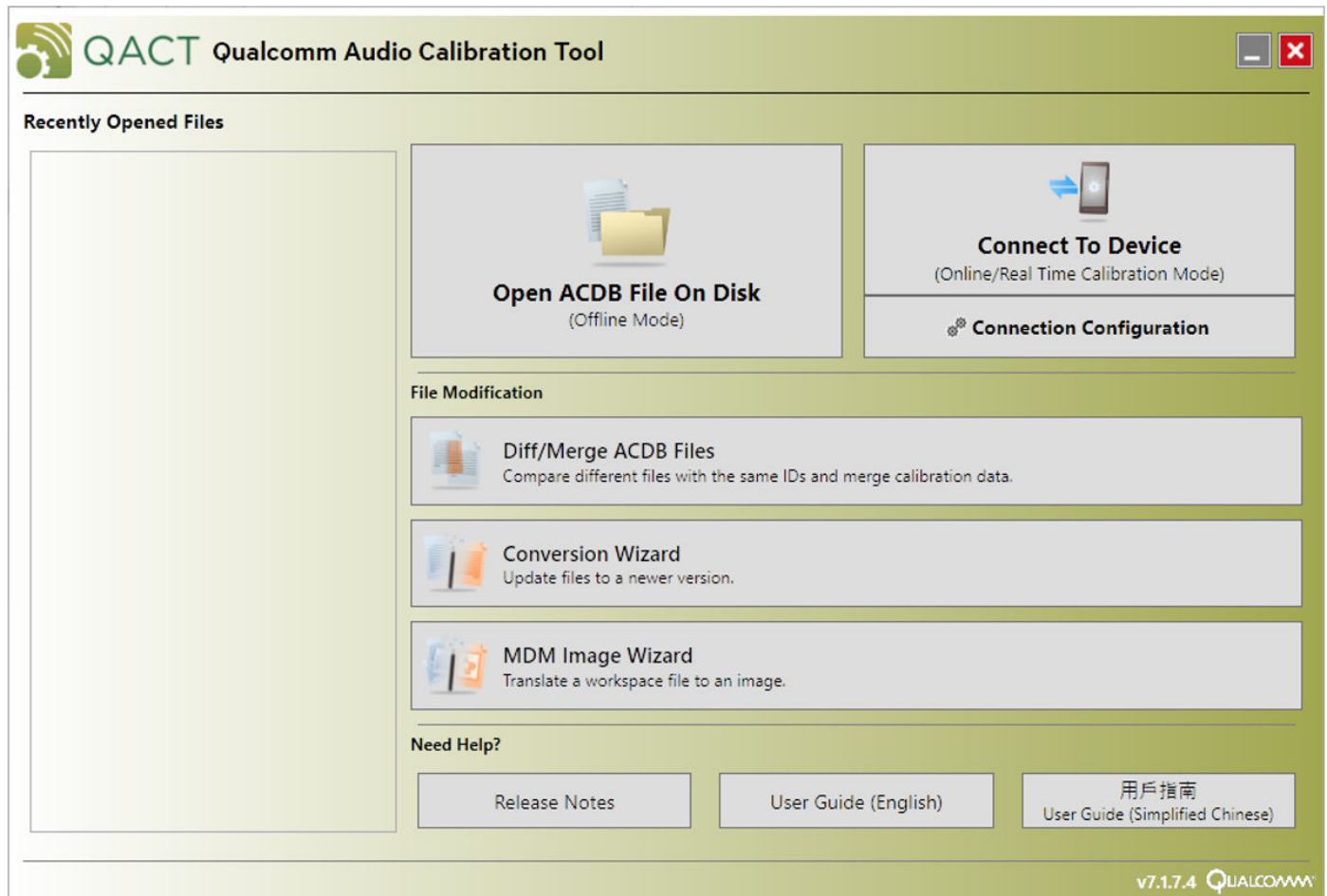


Figure 8-1 QACT User Interface

The QACT user interface includes:

- **Open ACDB File On Disk (offline mode)**
- **Connect To Device (online mode)**
- **Connection Configuration (online mode)**
- **Diff/Merge ACDB Files**
- **Conversion Wizard**
- **MDM Image Wizard**

8.1 Prerequisites for QACT tuning process

Before launching QACT, set up the QCC device for tuning by loading an ANC tuning case, as described in [ANC use cases](#). The device must be enumerated as a USB audio play and record device.

1. Check that the device exists and test it in the Windows sound control panel. On the **Playback** tab, right-click **USB audio device**. The audio device must be configured for 48000 Hz, 16-bit in the Windows sound properties.
2. Enable ANC by compile the Earbud Application with the option in the **Project Properties**. A valid ANC license must be present for both running and tuning ANC.

8.2 Connect the device to QACT

To run on the platform, QACT needs two file paths found within the Qualcomm ADK software, which are listed in [Table 8-1](#).

Table 8-1 QACT file paths

File	Description	Path
kalaccess.dll	Allows QACT to identify and communicate to the device.	<ADK path>/tools/bin/
QACT workspace file (*.qwsp).	Contains the parameter descriptions of all the audio processing capabilities running within the Kymera audio framework.	<ADK path>/audio/kalimba/Kymera/output/aura_rom_v02_release/QACT/*.qwsp or <ADK path>/audio \qcc514x_qcc304x \kalimba\kymera \output \streplus_rom_release\QACT *.qwsp

To connect a device to QACT:

1. Click the **Connection Configuration** link to verify that QACT has automatically populated an appropriate file. The `kalaccess.dll` file is located in the `<ADK path>/tools/bin/` folder.
2. Select the workspace file from following location: `<ADK path>/audio/kalimba/Kymera/output/aura_rom_v02_release/QACT/*.qwsp`
or
`<ADK path>/audio\qcc514x_qcc304x\kalimba\kymera\output \streplus_rom_release\QACT\*.qwsp`

3. Click **Done**, and then select **Connect to device**.

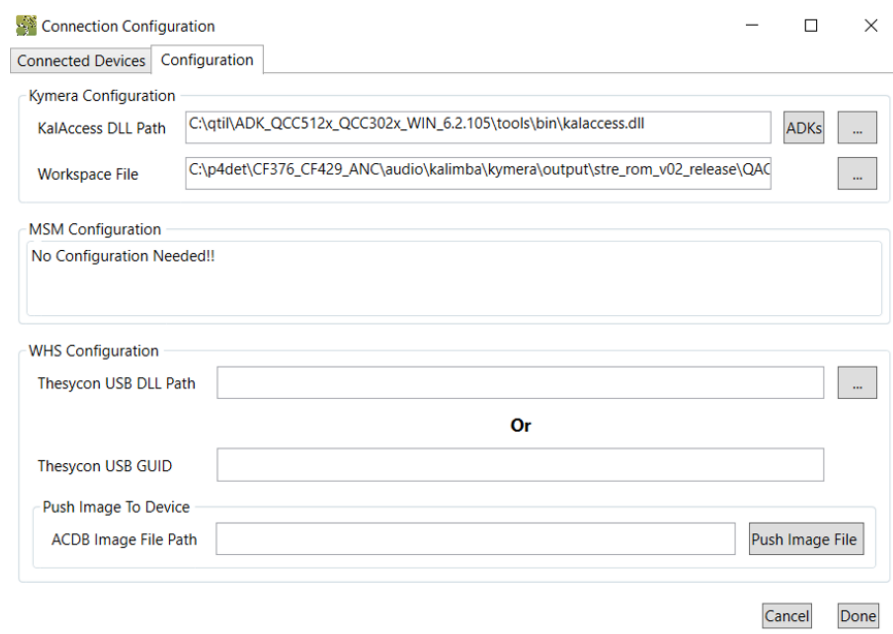


Figure 8-2 kalaccess.dll and workspace file path

4. Click the **(Proc0, SubSys 3)** link.

QACT recognizes the device. **SubSys 3** refers to the audio subsystem. **Proc 0** refers to the primary core within the audio subsystem.

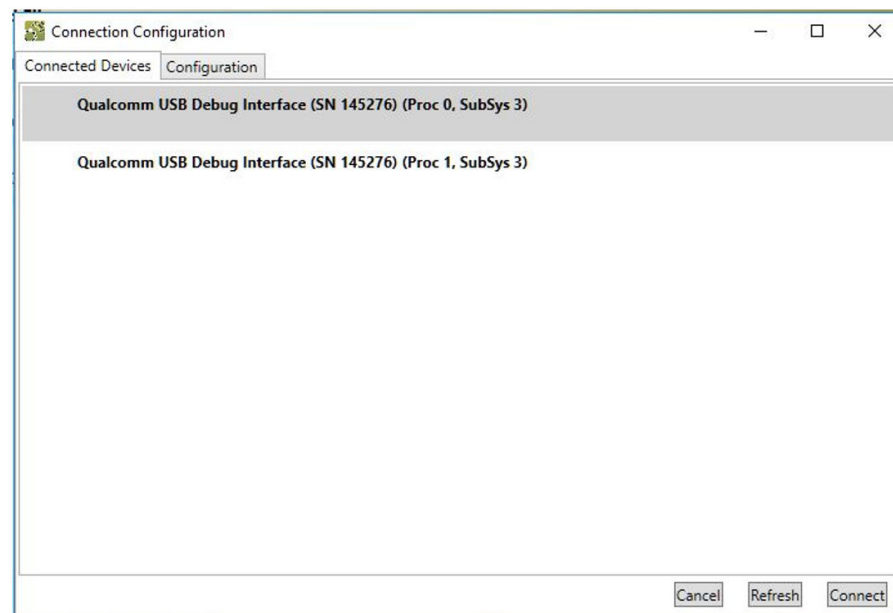


Figure 8-3 Connect device

5. Click **Connect**.
6. After the device is connected:
 - Generate and test an ANC design by using the simplified tuning workflow.
 - If you do not intend to use the simplified tuning workflow, populate ANC parameters manually with externally generated coefficients, by using the **Database** view.

8.3 Adaptive ANC tuning

QACT provides options to tune the adaptive ANC capability in either run mode or tuning mode.

The ANC tuning process involves the following tasks:

- Connecting headset mics and speakers to the development board
- Polarity test
- FxLMS and ED tuning

The tuning workflow involves only one ear cup at a time (left or right). The same filter is used on left and right ear cups in the final design. However, the tuning workflow supports each ear cup having independent filters, gains, and other parameters, if necessary.

NOTE From ADK 21.3 and onwards on QCC517x/QCC307x, functionality of Adaptive ANC between two capabilities (ANC Hardware Manager (AHM) and the Adaptive ANC Controller AANC2) is separated. The AHM capability takes all the gain control functionality from AANC, while the AANC2 capability retains the adaptive gain algorithms.

8.3.1 Prerequisites for tuning Adaptive ANC

Before tuning Adaptive ANC, configure the software build to enable Adaptive ANC. Adaptive ANC tuning mode is not required to tune the capability. Adaptive ANC tuning mode provides the ability to play/record information for offline analysis, but the tuning can be performed using QACT in either AANC tuning or run mode.

Additionally:

- Validate the microphone configuration and static ANC configurations.
- Connect the earbud to USBDBG on the table in a relatively quiet room, preferably away from a laptop fan.

To obtain graphs displayed in the tuning process, use the graphical logging tool mentioned in [DSP data logging environment for Adaptive ANC](#).

For more information, see *Qualcomm Kymera Capability Library User Guide*.

8.3.2 Preparing a new device for tuning

To tune a new device, activate Adaptive ANC by using the button, touch-press, GAIA, or Pydbg commands.

To prepare the device:

1. Open **QACT** and enable **statistics**.

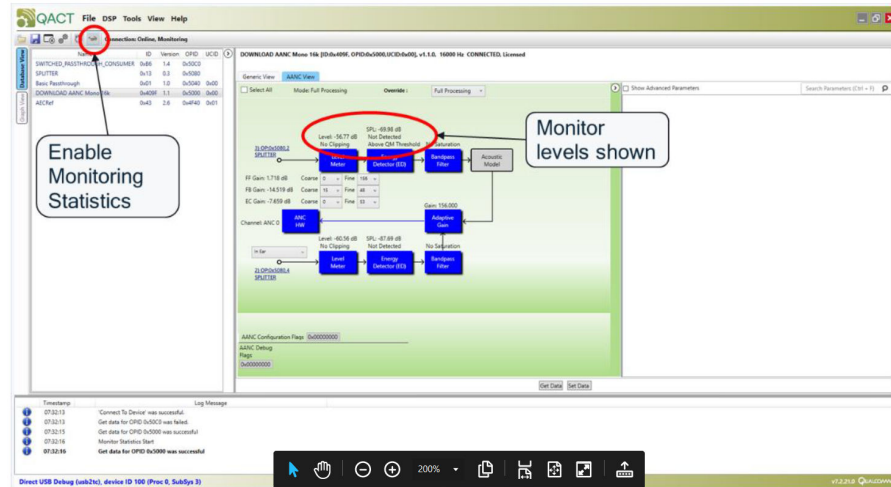


Figure 8-4 Enable statistics in QACT

2. Disable the energy detection:

- a. Disable all energy detectors and self-speech detection by setting the AANC Configuration Flags to bypass all values (0xF).

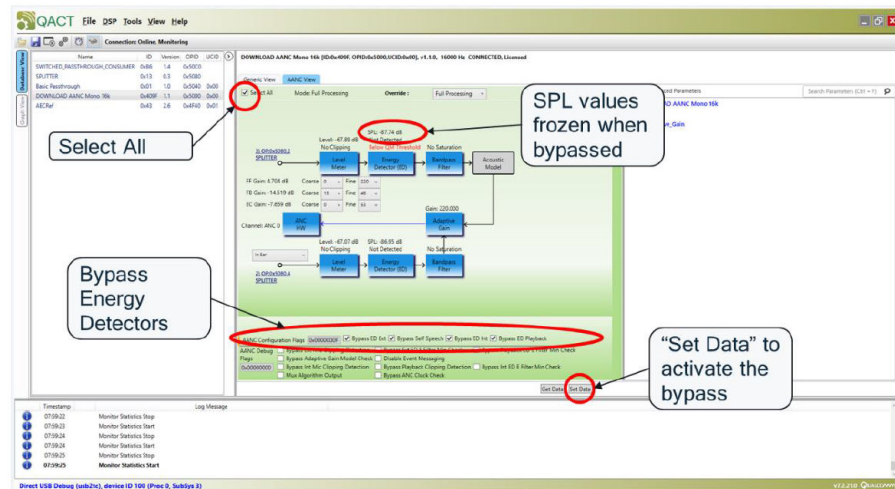


Figure 8-5 Disable energy detectors in QACT

3. Conduct the polarity test by observing the Adaptive gain statistic value AG_CALC:

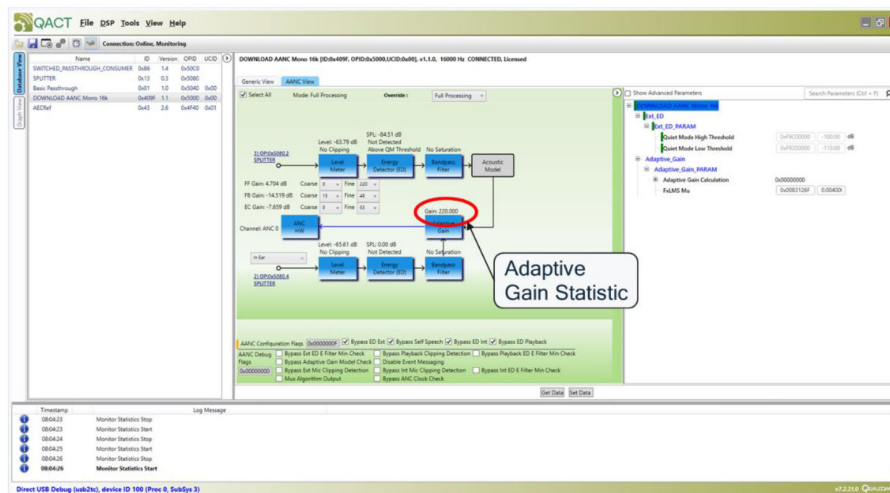


Figure 8-6 Adaptive gain statistic displayed in QACT

- If Adaptive gain statistic is 220 (or max gain bound):
 - Occlude the earbud with your finger and check the direction of Adaptive gain.
 - If the Adaptive gain value is coming down from 220 to a different value, the polarity is good and the device is ready for further tuning.
- If Adaptive gain statistic is 40 (or min gain bound):
 - Change the value of $FxLMS \mu_u$ to $-FxLMS \mu_u$, and then try again.
 - This step is needed to understand the polarity which provides ANC.
 - The static ANC filter design process previously completed will incorporate speaker polarity assumptions into the IIR coefficients.

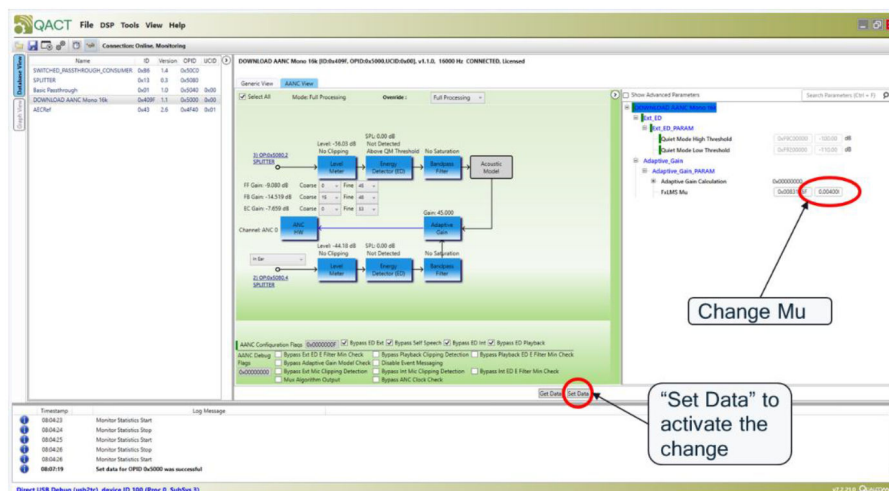


Figure 8-7 Change FxLMS Mu parameter in QACT

- If Adaptive gain statistic is 220 (or max gain bound):
 - Occlude the earbud with your finger and see the direction of adaptive gain.
 - If the Adaptive gain value is coming down from 220 to a different value, the device is ready for further tuning.

If Adaptive gain statistic is stuck at 220 (or max gain bound), you need to change to the STATIC system mode using the mode override, change the coarse gain to one above current value, disable the override and repeat the test.

NOTE When tuning AANC2, the gain override applies to the AHM. Switch the AHM to static mode, change the coarse gain, and then disable the override.

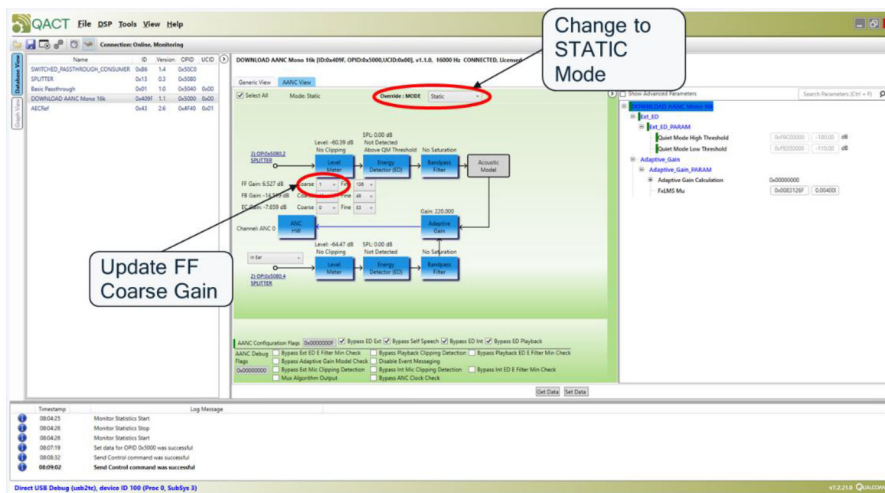


Figure 8-8 Change the coarse gain in static mode from QACT

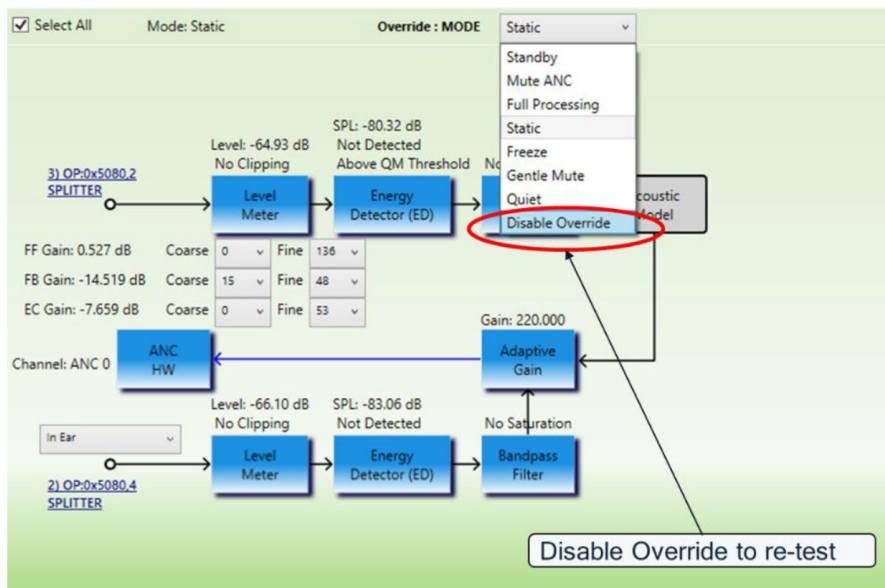


Figure 8-9 Disable mode override to re-test tuning in QACT

4. Proceed to FxLMS tuning. For more information, see [FxLMS tuning: Configure Convergence speed and Converged value](#). Use the provided default settings as a starting point.

8.3.3 Tunable parameters

ANC includes various parameters that can be tuned using QACT.

AANC_CONFIG (AANC Configuration Flags)

Default: 0x0

Configuration bitflags to disable energy detectors. This can be useful when tuning the FxLMS. In QACT, these are displayed below the diagram in the AANC View (see Figure 8-10) or available as a drop-down in the Generic View under the AANC Configuration Flags parameter.

- 0x1: Disable Internal ED
- 0x2: Disable External ED
- 0x4: Disable Playback ED
- 0x8: Disable Self-Speech detection

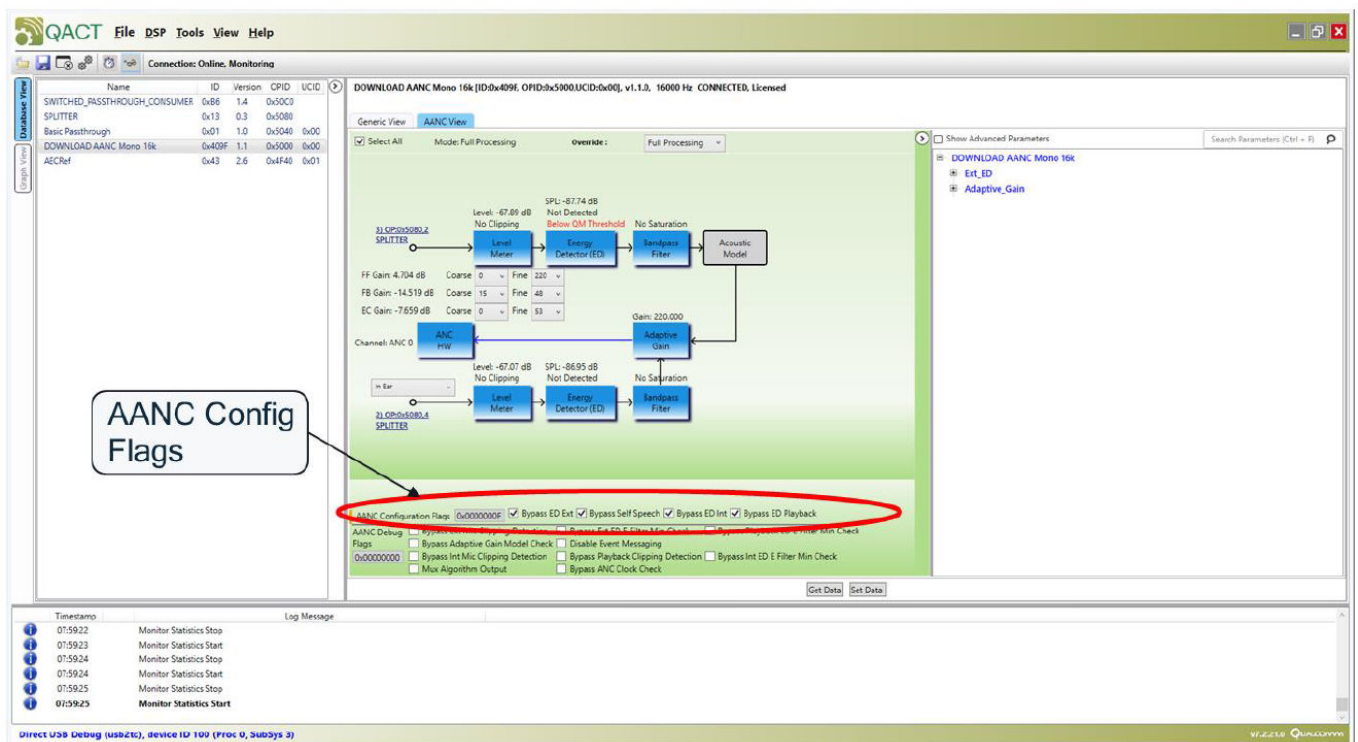


Figure 8-10 AANC CONFIG flags displayed in QACT

DISABLE_AG_CALC (Adaptive Gain Calculation)

Default: 0 (Boolean)

Single bit to disable the adaptive gain calculation

NOTE Not used in AANC2 (Change SYSMODE to FREEZE instead if the Adaptive gain algorithm needs to be disabled).

FXLMS_MU (FxLMS Mu)

Default: 0.004

Adaptation rate for the FxLMS algorithm. Higher values correspond to faster convergence, at the expense of increased gain variance.

GENTLE_MUTE_TIMER (Gentle Mute Timer)

Default: 0.1 (s)

Amount of time it takes for the fine gain to ramp down from current value to zero when the operator is in Gentle Mute mode. This helps provide a seamless listening experience during use-case transitions.

NOTE This parameter has moved to the AHM in AANC2.

AANC_DEBUG (AANC Debug Flags)

Default: 0x0

Debug bitflags for various blocks within the operator. Of particular use in the tuning procedure are the option to disable the ear sensor check and the option to send post-filter data to the output buffers.

- 0x0001: Disable the model check that ensures the operator has a model loaded prior to starting.
- 0x0002: Disable the ear sensor check. (Not in AANC2 - moved to AHM)
- 0x0004: Disable the ANC HW clock check. (Not in AANC2 - moved to AHM)
- 0x0100: Select the output terminal buffer contents (0 is passthrough, 1 is output after filters).
- 0x1000: Disable clipping detection on the internal microphone.

- 0X2000: Disable clipping detection on the external microphone.
- 0X4000: Disable clipping detection on the playback stream.

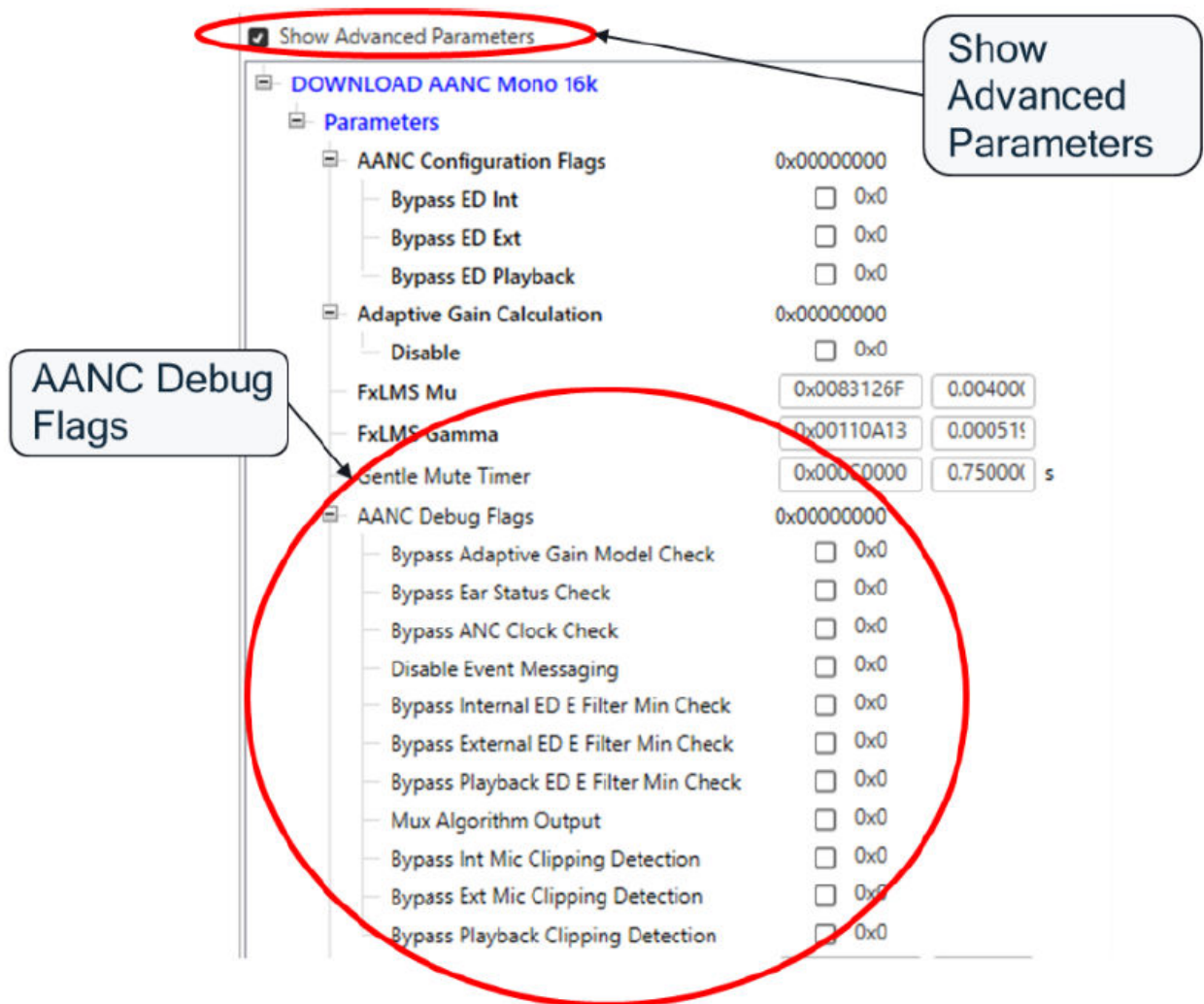


Figure 8-11 AANC debug configuration flags displayed in the QACT generic view

QUIET_MODE_HI_THRESHOLD (Quiet Mode High Threshold)

Default: -70 (dBFS)

Quiet mode high threshold in dBFS. When the external noise is louder than this value for EVENT_QUIET_CLEAR seconds, a message to exit quiet mode is sent by the operator.

QUIET_MODE_LO_THRESHOLD (Quiet Mode Low Threshold)

Default: -80 (dBFS)

Quiet mode low threshold in dBFS. When the external noise loudness is below this value for EVENT_QUIET_DETECT seconds, a message to enter quiet mode is sent by the operator.

ED_EXT_ENVELOPE (External ED Envelope)

Default: 0.03 (s)

Specifies the time constant for the stationary noise estimate. Smaller values elicit a faster response, at the expense of a less accurate estimate of the stationary noise.

ED_EXT_MIN_SIGNAL (External ED Minimum Signal)

Default: 15 (dBFS)

Specifies the threshold that differentiates stationary noise from non-stationary noise. Increasing this value reduces the sensitivity to non-stationary noise. Decreasing it makes the energy detector trigger more often.

ED_INT_ENVELOPE (Internal ED Envelope)

Default: 0.009 (s)

Specifies the time constant for the stationary noise estimate. Smaller values elicit a faster response, at the expense of a less accurate estimate of the stationary noise.

ED_INT_MIN_SIGNAL (Internal ED Minimum Signal)

Default: 15 (dBFS)

Specifies the threshold that differentiates stationary noise from non-stationary noise. Increasing this value reduces the sensitivity to non-stationary noise. Decreasing it makes the energy detector trigger more often.

ED_PB_ENVELOPE (Playback ED Envelope)

Default: 0.03 (s)

Specifies the time constant for the stationary noise estimate. Smaller values elicit a faster response, at the expense of a less accurate estimate of the stationary noise.

ED_PB_MIN_SIGNAL (Playback ED Minimum Signal)

Default: 8 (dBFS)

Threshold used to differentiate stationary noise from non-stationary. Increasing this value reduces the sensitivity. Decreasing it makes the energy detector trigger more often.

SELF_SPEECH_THRESHOLD (Self Speech Threshold)

Default: 20 (dBFS)

Sensitivity of the self-speech detector: higher values mean less sensitivity.

FXLMS_LAMBDA (FxLMS Lambda)

Default: 1.0

Fine tuning parameter for FxLMS sweet spot.

TARGET_NOISE_REDUCTION (Target Noise Reduction)

Default: 0.0

Control the amount of noise reduction that Adaptive ANC performs. Value between 0.0 (maximum reduction) and 1.0 (no reduction).

QUIET_MODE_TIMER (Quiet Mode Timer)

Default: 0.1 (s)

Amount of time it takes for the fine gain to ramp down from current value to zero when the operator is in Quiet mode.

NOTE This parameter has moved to the AHM in AANC2.

FF_FINE_RAMP_UP_TIMER (FF Fine Gain Ramp Timer)

Default: 0.1 (s)

Amount of time it take for the FF fine gain to ramp up to static value (STATIC mode) or FXLMS_INITIAL_VALUE (FULL processing mode) when switching to either of these modes.

NOTE This parameter has moved to the AHM in AANC2.

FB_FINE_RAMP_UP_TIMER (FB Fine Gain Ramp Timer)

Default: 0.1 (s)

Amount of time it take for the FB fine gain to ramp up to static value when switching to either STATIC or FULL modes.

NOTE This parameter has moved to the AHM in AANC2.

FB_FINE_RAMP_DELAY_TIMER (FB Fine Gain Ramp Delay Timer)

Default: 0.0 (s)

Additional delay between starting adaptive ANC and ramping the FB fine gain to its static value. This can be used to reduce pops.

NOTE This parameter has moved to the AHM in AANC2.

Optional tunable parameters

Following tunable parameters are optional:

ED_EXT_RATIO (External ED Ratio)

Default: 0.55

Ratio of stationary to non-stationary loudness. Lower values increase the sensitivity of the energy detector.

ED_EXT_DELTA_TH (External ED Delta Threshold)

Default: 2 (dBFS)

Dynamic between maximum power and power to avoid ramping in stationary noise.

ED_EXT_E_FILTER_MIN_THRESHOLD (External ED E Filter Min Threshold)

Default: 5 (dBFS)

Helps the stationary noise estimate to become unstuck in the presence of sudden increase in the background noise (notice mostly in testing environments). If after

ED_EXT_E_FILTER_MIN_COUNTER_THRESHOLD frames the stationary noise estimate is
ED_EXT_E_FILTER_MIN_THRESHOLD dB away from the power estimate, it is reset.

ED_EXT_E_FILTER_MIN_COUNTER_THRESHOLD (External ED E Filter Min Count Threshold)

Default: 250 (frames)

Used in conjunction with ED_EXT_E_FILTER_MIN_THRESHOLD denotes the number of frames the stationary noise estimate needs to be away from the power estimate to reset it.

ED_INT_RATIO (Internal ED Ratio)

Default: 0.45

Ratio of stationary to non-stationary loudness. Lower values increase the sensitivity of the energy detector.

ED_INT_DELTA_TH (Internal ED Delta Threshold)

Default: 5 (dBFS)

Dynamic between maximum power and power to avoid ramping in stationary noise.

ED_INT_E_FILTER_MIN_THRESHOLD (Internal ED E Filter Min Threshold)

Default: 5 (dBFS)

Helps the stationary noise estimate to become unstuck in the presence of sudden increase in the background noise (notice mostly in testing environments). If after

ED_INT_E_FILTER_MIN_COUNTER_THRESHOLD frames the stationary noise estimate is
ED_INT_E_FILTER_MIN_THRESHOLD dB away from the power estimate, it is reset.

ED_INT_E_FILTER_MIN_COUNTER_THRESHOLD (Internal ED E Filter Min Count Threshold)

Default: 250

(Frames) Used in conjunction with ED_INT_E_FILTER_MIN_THRESHOLD denotes the number of frames the stationary noise estimate needs to be away from the power estimate in to reset it.

ED_PB_RATIO (Playback ED Ratio)

Default: 0.55

Ratio of stationary to non-stationary loudness. Lower values increase the sensitivity of the energy detector.

ED_PB_DELTA_TH (Playback ED Delta Threshold)

Default: 2 (dBFS)

Dynamic between maximum power and power to avoid ramping in stationary noise.

ED_PB_E_FILTER_MIN_THRESHOLD (Playback ED E Filter Min Threshold)

Default: 5 (dBFS)

Helps the stationary noise estimate to become unstuck in the presence of sudden increase in the background noise (notice mostly in testing environments). If after

ED_PB_E_FILTER_MIN_COUNTER_THRESHOLD frames the stationary noise estimate is
ED_PB_E_FILTER_MIN_THRESHOLD dB away from the power estimate, it is reset.

ED_PB_E_FILTER_MIN_COUNTER_THRESHOLD (Playback ED E Filter Min Count Threshold)

Default: 250 (Frames)

Used in conjunction with ED_PB_E_FILTER_MIN_THRESHOLD denotes the number of frames the stationary noise estimate needs to be away from the power estimate to reset it.

Bandpass filters

The AANC capability has a bandpass filter on each microphone input to control the bandwidth over which the operator will respond to changes in the noise environment. By default this is between 50Hz and 400Hz.

Bandpass filter: It is recommended to use Matlab's function butter() to design the filter. For example, the default filter is a 2nd order butterworth between 80 Hz and 400 Hz:

```
>> [bpf_num,bpf_den]: butter(2,[80 400]/8e3);
```

```
>> bpf_num_scaled: bpf_num/8;
```

```
>> bpf_den_scaled: bpf_den/8;
```

NOTE In AANC2, the format of these parameters has changed from Q1.N to Q4.N. This means, the divide by 8 step can be omitted and the coefficients can be directly used.

Types of BandPass filters:

- **BPF_DENOMINATOR_COEFF_EXT_nnn (BPF external a)**

Stores the denominator of the FxLMS bandpass filter for the external signal. The format is [a0, a1, a2, a3, a4]. The algorithm expects for the coefficients to be scaled by 1/8, so the a0 value will always be 0.125.

NOTE In AANC2, the format of these parameters has changed from Q1.N to Q4.N. This means, the divide by 8 step can be omitted and the coefficients can be directly used.

- **BPF_NUMERATOR_COEFF_EXT_nnn (BPF external b)**

Stores the numerator of the FxLMS bandpass filter for the internal signal. The format is [b0, b1, b2, b3, b4]. The algorithm expects for the coefficients to be scaled by 1/8.

NOTE In AANC2, the format of these parameters has changed from Q1.N to Q4.N. This means, the divide by 8 step can be omitted and the coefficients can be directly used.

- **BPF_DENOMINATOR_COEFF_INT_nnn (BPF internal a)**
Stores the denominator of the FxLMS bandpass filter for the external signal. The format is [a0, a1, a2, a3, a4]. The algorithm expects for the coefficients to be scaled by 1/8, so the a0 value will always be 0.125.

NOTE In AANC2, the format of these parameters has changed from Q1.N to Q4.N. This means, the divide by 8 step can be omitted and the coefficients can be directly used.
- **BPF_NUMERATOR_COEFF_INT_nnn (BPF internal b)**
Stores the numerator of the FxLMS bandpass filter for the internal signal. The format is [b0, b1, b2, b3, b4]. The algorithm expects for the coefficients to be scaled by 1/8.

NOTE In AANC2, the format of these parameters has changed from Q1.N to Q4.N. This means, the divide by 8 step can be omitted and the coefficients can be directly used.

CLIPPING_DURATION_EXT (Ext Mic Clip Duration)

Default: 0.5 (s)

Amount of time in seconds, to freeze adaptation when clipping is detected in the external microphone.

CLIPPING_DURATION_INT (Int Mic Clip Duration)

Default: 0.5 (s)

Amount of time, in seconds, to freeze adaptation when clipping is detected in the internal microphone.

CLIPPING_DURATION_PB (Playback Clip Duration)

Default: 0.5

Amount of time in seconds, to freeze adaptation when clipping is detected in the playback signal.

FXLMS_MIN_BOUND (FxLMS Min Bound)

Default: 40 (Steps)

Minimum bound for the FxLMS gain.

FXLMS_MAX_BOUND (FxLMS Max Bound)

Default: 220 (Steps)

Maximum bound for the FxLMS gain.

FXLMS_INITIAL_VALUE (FxLMS Initial Value)

Default: 40 (Steps)

Initial value for the FxLMS gain.

- NOTE** This parameter has been removed from AANC2. When AANC2 is set to FULL mode, it uses whatever the current gain value is as the start point for the FxLMS algorithm.

EVENT_GAIN_STUCK (Hold Time: Unchanged Gain Event)

Default: 5 (s)

Amount of time in seconds, to detect when the gain has been stuck (for debugging). After this time a trigger message is sent by the operator.

EVENT_ED_STUCK (Hold Time: Unchanged ED Event)

Default: 5 (s)

Amount of time in seconds, to detect when the energy detector has been stuck (for debugging). After this time a trigger message is sent by the operator.

EVENT_QUIET_DETECT (Hold Time: Quiet Mode Detected)

Default: 15 (s)

Amount of time in seconds, that the loudness estimate of the external noise has to be below `QUIET_MODE_LO_THRESHOLD` to trigger quiet mode. After this time a trigger message is sent by the operator.

EVENT_QUIET_CLEAR (Hold Time: Quiet Mode Cleared)

Default: 5 (s)

Amount of time in seconds, that the loudness estimate of the external noise has to be above `QUIET_MODE_HI_THRESHOLD` to clear quiet mode. After this time a clear trigger message is sent by the operator.

EVENT_CLIP_STUCK (Hold Time: Unchanged Clip Event)

Default: 5 (s)

Amount of time in seconds, to detect when the clipping flag has been stuck (for debugging). After this time a trigger message is sent by the operator.

EVENT_SAT_STUCK (Hold Time: Unchanged Saturation Event)

Default: 5 (s)

Amount of time in seconds, to detect when the saturation flag has been stuck (for debugging). After this time a trigger message is sent by the operator.

EVENT_SELF_TALK (Hold Time: Self-Talk Event)

Default: 5 (s)

Amount of time in seconds, to detect when the internal mic level is higher than the external mic level has been stuck (for debugging). After this time a trigger message is sent by the operator.

EVENT_SPL (Hold Time: SPL Event)

Default: 0.5 (s)

Amount of time in seconds, to detect when the external microphone signal is above a threshold controlled by `EVENT_SPL_THRESHOLD`. After this time, a trigger message is sent by the operator. When the level drops below this threshold, a negative trigger message is immediately sent.

EVENT_SPL_THRESHOLD (SPL Event Threshold)

Default: -6 (dBFs)

Threshold above which the SPL Event will be triggered.

8.3.4 FxLMS tuning: Configure Convergence speed and Converged value

The performance of the Adaptive algorithm depends on the ANC filters and on the signal from the internal microphone. Therefore, the most important part is the design of the feed-forward and feedback controllers.

For the adaptive algorithm, there are two features that are configurable by the end user:

- **Convergence speed:** Controls how fast or slow the algorithm converges the gain for optimal performance. The convergence speed is independent of the type of noise of the loudness level, but small variations can be noticed.
- **Converged value:** For some designs, it is possible for the value reached by the algorithm to be a little off from the optimum solution; in this case, a parameter is provided for fine tuning. However, the user must be sure that this fine tuning applies to a large majority of costumers.

NOTE For more information about parameters referenced in this section, see [Tunable parameters](#).

Convergence speed

The speed to which the algorithm converges to a solution affects not only the time it takes to arrive to the optimum value, but also how much can the optimum value vary once it has converged. Slower

speeds mean low variance, high speeds high variance. If the variance is too high it will have the undesirable effect of being audible to the user as continually changing ANC performance.

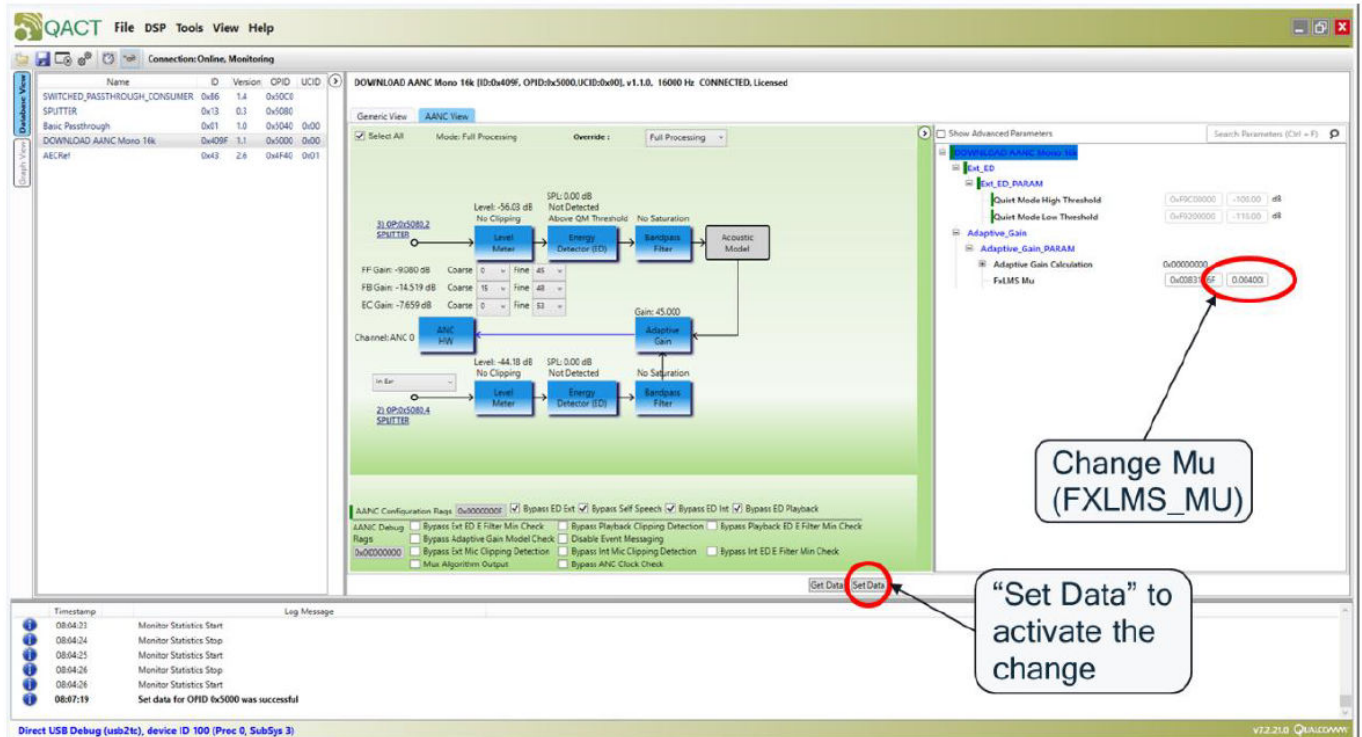


Figure 8-12 Updating FxLMS Mu parameter in QACT

The parameter FXLMS_MU tunes the speed. The default value is 0.004. The following plot shows an example of convergence from the maximum gain to the optimum value. After the algorithm has converged, the variance of the gain is low.

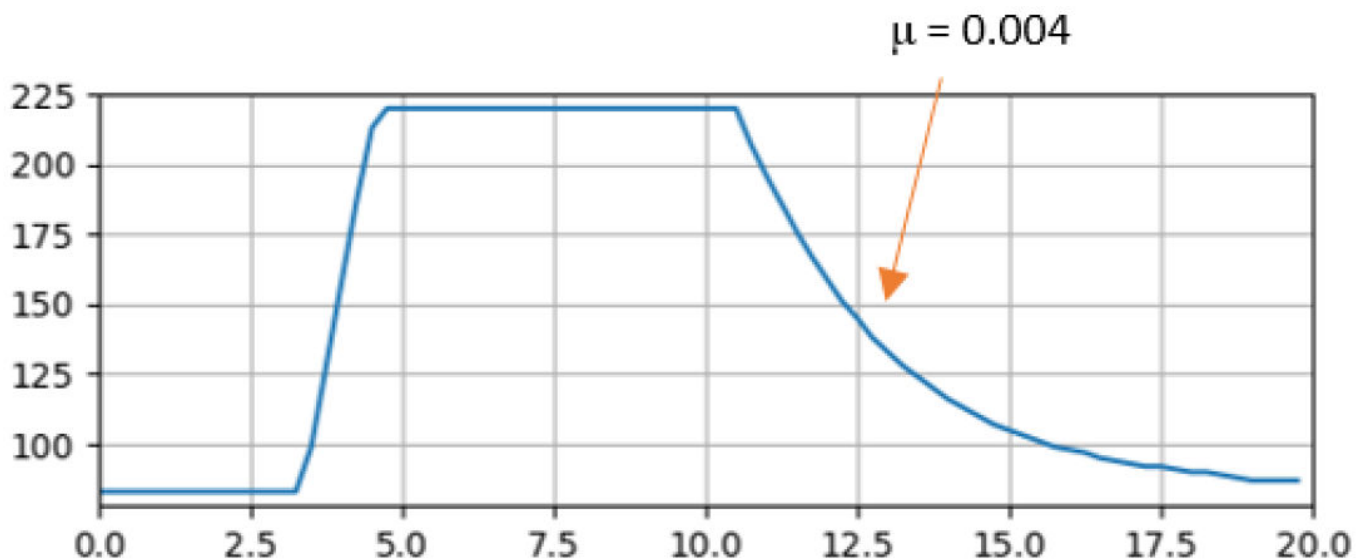


Figure 8-13 Gain convergence with default value

Increasing the parameter FXLMS_MU to 0.04 (10x increase) produces a much faster convergence. After the algorithm has converged, the variance of the gain is high.

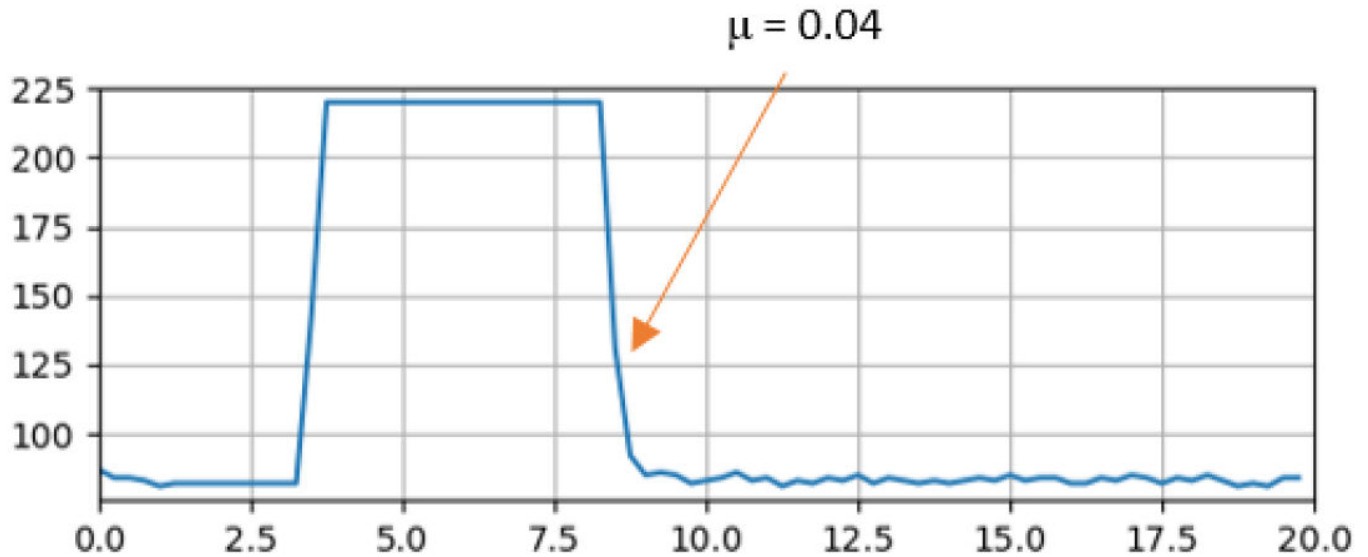


Figure 8-14 Gain convergence with increased FXLMS_MU value

Converged value

The Adaptive algorithm tries to converge to the best solution based on the error signal. However, there can be a difference between the signal perceived at the error (feedback) microphone and the one perceived by the listener. If this is the case, it is possible to nudge the converged value towards the desired solution by using the parameter FXLMS_LAMBDA. By default, it is set to 1.0.

If a higher gain is desired a small increase is used, for example FXLMS_LAMBDA: 1.05. If a lower gain is what is required, a small decrease should be used, but this case seldom happens; it has been

noticed that slightly higher gains might be required, not smaller. An example would be FXLMS_LAMBDA: 0.95.

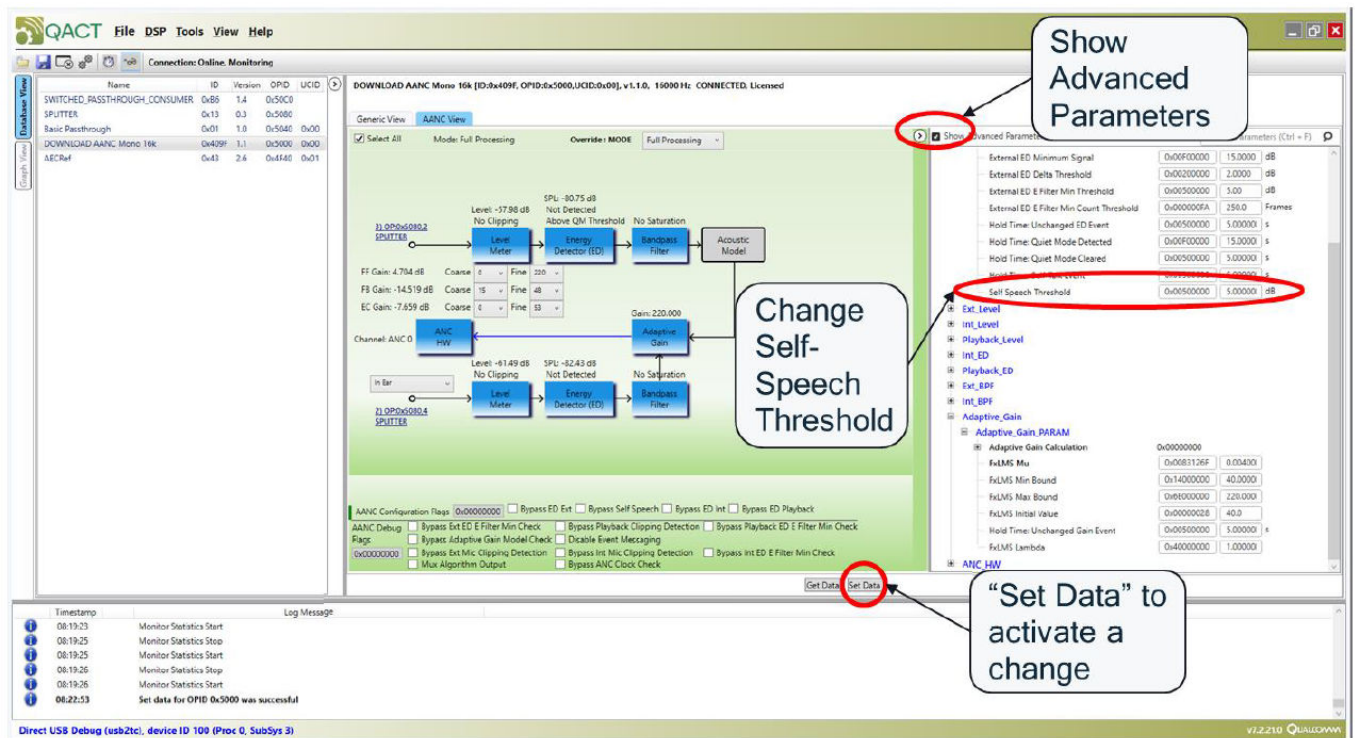


Figure 8-15 Updating FxLMS Lambda parameter in QACT

Figure 8-16 shows the gain after converging to 85 with FXLMS_LAMBDA: 1.0.

If FXLMS_LAMBDA is set to 1.05 with the corresponding change in the converged gain, change the FXLMS_LAMBDA value to 0.96 and at the end, it is changed back to the default value of FXLMS_LAMBDA: 1.0, where the gain converges back to the original value.

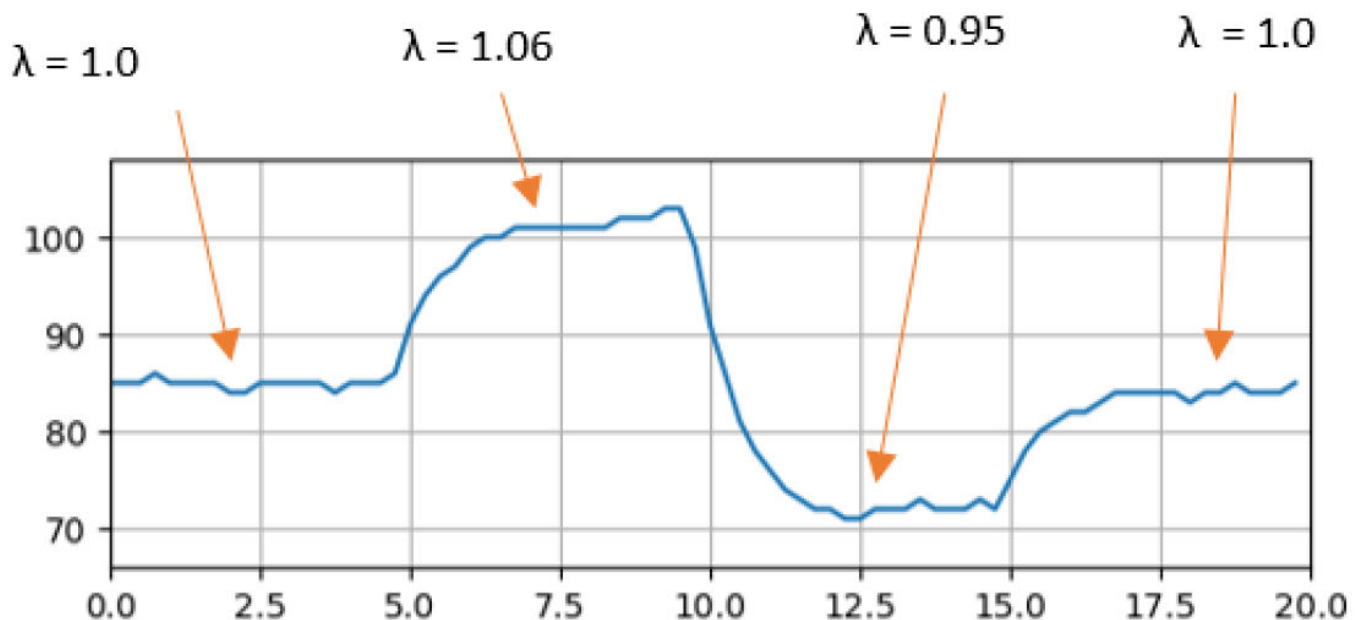


Figure 8-16 Gain converge variation with FXLMS_MU value

8.3.5 Tuning Adaptive ANC when the adaptation rate is not optimum

The adaptation rate is not considered to be optimum when either the adaptation rate is too slow or too fast, or when the converged value overshoots and undershoots the desired solution.

Tuning when adaptation rate too slow/fast

Depending on the product design, either a slow convergence that showcases the algorithm performing the adaptation can be chosen, or a fast convergence to make the adaptation invisible to the user. Set the parameter FXLMS_MU to the desired value, taking into account that the variance of the converged gain will be affected.

The best way to measure the speed of convergence is by either plotting the gain in real-time, or listening to the noise attenuation, when transitioning from out-of-ear state to in-ear state. This assumes that the out-of-ear detector has been disabled.

1. Wear the earbud.
2. Disable out-of-ear detector.
3. Play environmental stationary noise.
4. Wait for the algorithm to converge.
5. Remove earbud from ear and wait for it to converge to the maximum allowed gain.
6. While looking at the gain, put the earbud back in the ear.
7. The Energy detector might trigger in this case, freezing the adaptation for a moment, so it could take some time for the gain to start adapting.

8. Wait until it converges to 90% of the optimum value.
9. Measure the amount of time taken.

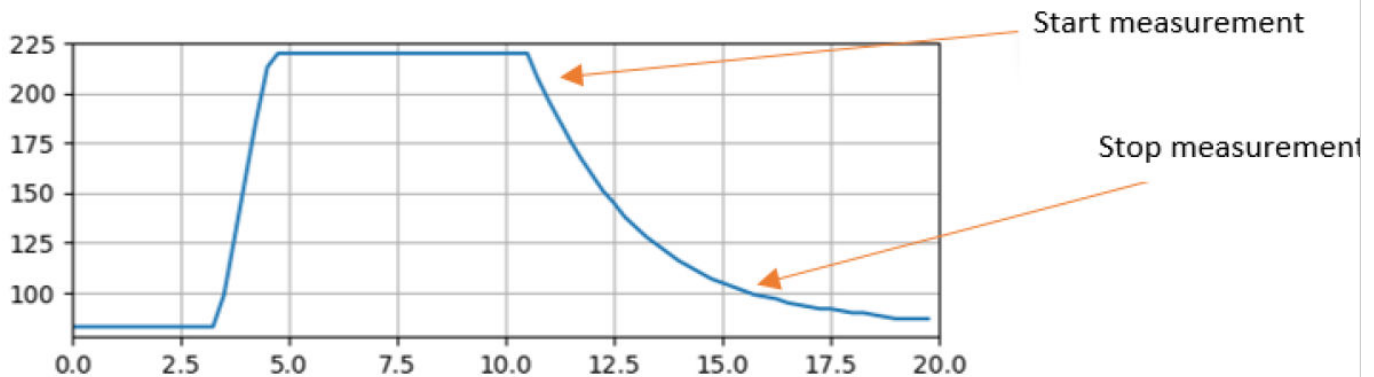


Figure 8-17 Measuring rate of gain convergence

Converged value overshoots/undershoots desired solution

During listening tests, when going from out-of-ear to in-ear, or by artificially changing the current gain, while the algorithm converges, it is possible to perceive a point where the attenuation is preferred right before the one obtained when convergence happens. If this is the case, it is possible to nudge the gain to the desired sweet spot.

1. Wear the earbud.
2. Disable out-of-ear detector.
3. Play environmental stationary noise.
4. Wait for the algorithm to converge.
5. Remove earbud from ear and wait for it to converge to the maximum allowed gain. Alternatively, the current gain can be changed manually to the maximum value. If this is the case, there is no need to remove the earbud.
6. Listen to the attenuation and identify if there is a sweet spot in the way to the optimum gain, while convergence happens. A slow adaptation rate is recommended.
7. If a sweet spot is identified, increase the FXLMS_LAMBDA parameter by a small amount and repeat the test, until the desired value is obtained.

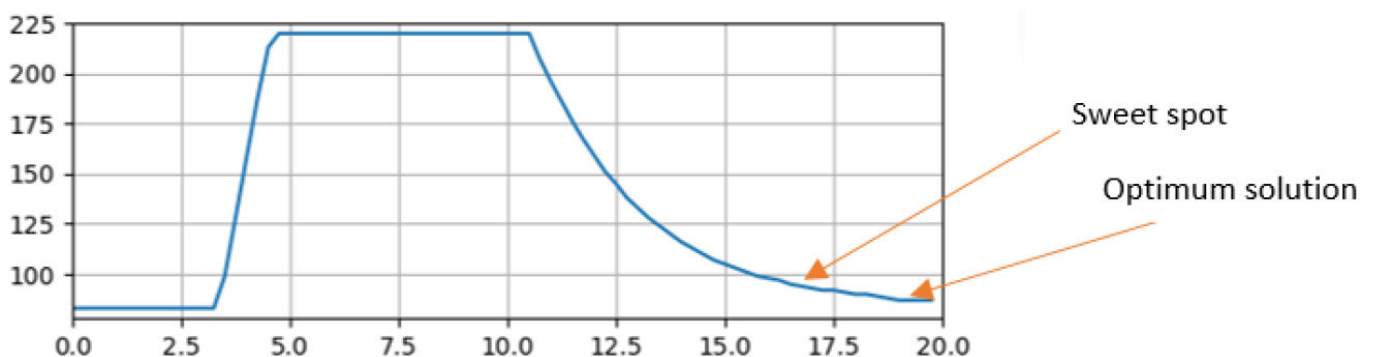


Figure 8-18 Finding sweet spot for gain convergence

8.4 Adaptive Ambient tuning

QACT provides a way to tune the capabilities involved in the Adaptive Ambient graph in run mode.

The adaptive ambient tuning process involves the following tasks:

- Connecting headset mics and speakers to the development board.
- ANC Comander (ADRC) tuning.
- Howling Control and Gain Recovery (HCGR) tuning.

The tuning workflow involves only one ear cup at a time (left or right). Usually, the same filter is used on left and right ear cups in the final design. However, the tuning workflow supports each ear cup having independent filters, gains, and other parameters, if necessary.

Prerequisites for Adaptive Ambient tuning

- Configure software build to enable Adaptive Ambient.
- Valid microphone configuration and static ANC configurations.
- Earbud connected to USBDBG on the table in a relatively quiet room, preferably away from a laptop fan.

Any graphs displayed in the tuning process can be obtained by using the graphical logging tool. The parameters described in the following sections are detailed both in the *Kymera Capability Library User Guide*.

ANC Comander tuning

The ANC Comander (ADRC) controls the ANC feed-forward gain to perform compression and expansion on the external microphone signal in an Adaptive Ambient (AAMB) graph. For more information, see *Qualcomm Kymera Capability Library User Guide*.

The ANC Comander can be tuned following a similar procedure to a regular compander, and should be tuned to meet the desired performance for the target acoustics.

Five thresholds can be set with different gain ratios and knee widths. The level estimation can be switched between peak and RMS. For more information, see the *Compander* section in the *Kymera Capability User Guide*.

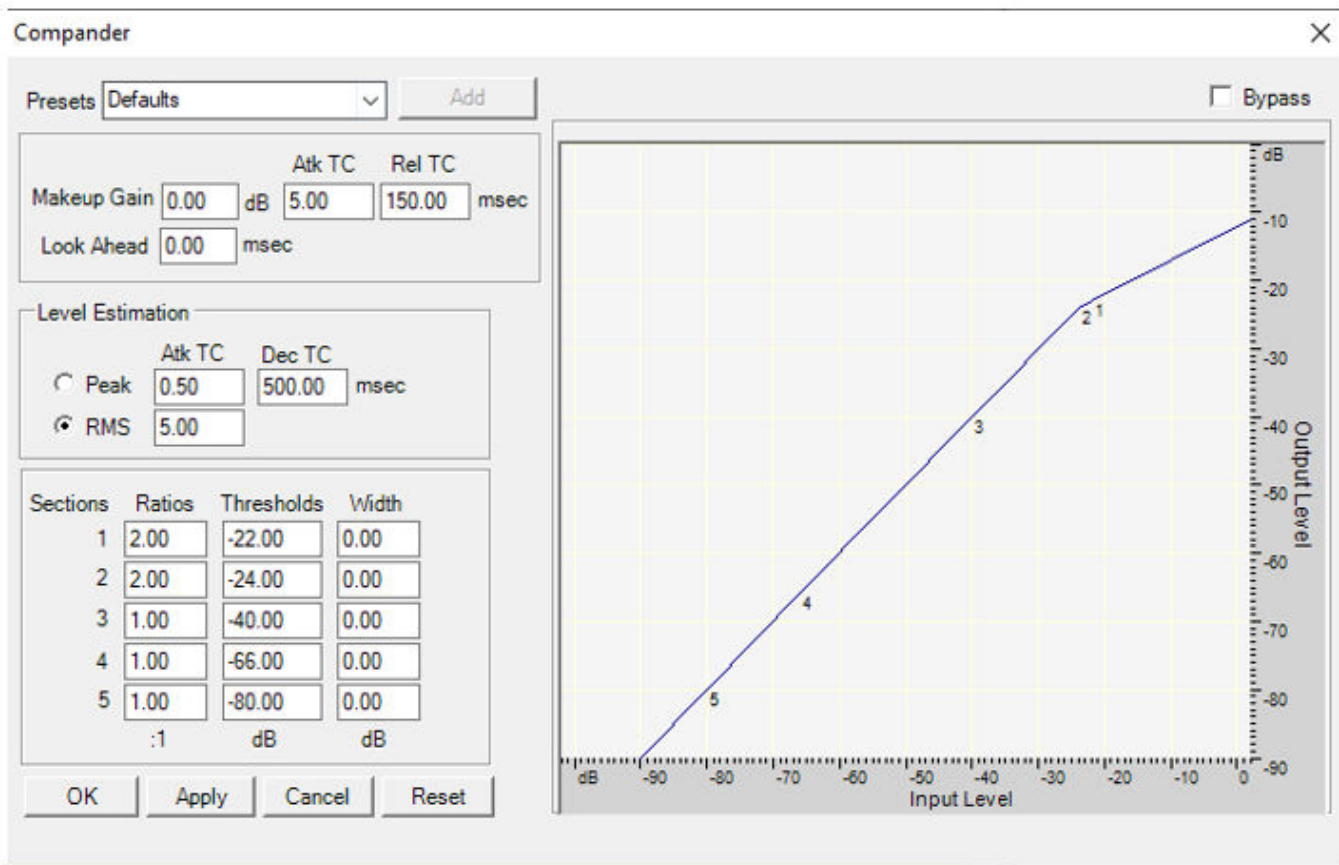


Figure 8-19 ANC Compander tuning interface

Howling Control and Gain Recovery tuning

The Howling Control and Gain Recovery (HCGR) operator provides the following two functions:

- Detect the presence of howling.
- Mitigate howling by reducing the gain and recovering to the normal operating level.

Detecting howling

The howling detection algorithm operates in the frequency domain with the following steps:

1. Peak to threshold power ratio (PTPR).

- a. Compares the power of each bin to a threshold, marking any bin greater than the threshold as potential howling.
- b. Parameters: `PTPR Threshold`, `PTPR Threshold Bexp`. The thresholds (0-1) is scaled by the frame component (Bexp) as $\text{Threshold} * 2^{-(\text{PTPR THRESHOLD Bexp})}$.

PTPR Threshold Bexp	Power scaling	Power scaling
0	1	0
1	2-1	-6
2	2-2	-12
-	-	-
n	2-n	-(6*n)

- c. Keep the mantissa (`PTPR Threshold`) fixed and tune the exponent.
- d. `PTPR Threshold DC Bexp` is provided as a separate configurable value for the power in the DC bin.
- e. Example: 1.0 (mantissa) and 5 (exponent) mean that signals must be greater than -30 dBFS to qualify as howling.

2. Peak to average power ratio (PAPR)

- a. Compares the power of potential howling bins with the average power.
- b. Parameter: `PAPR Threshold Shift`. The average power is shifted by this parameter to provide the threshold against which the bin power is compared. This is the equivalent of multiplication by 2^{shift} .
- c. Example: A shift value of 8 means that a bin is considered to have a tone if its power is 8 times that of the average signal power in the frame.

3. Peak to neighbour power ratio (PNPR)

- a. Compares the power of potential howling bins with that of their neighbours.
- b. Where possible this comparison is against the second neighbour rather than the immediate to avoid spectral leakage issues.
- c. Parameter: `PNPR Threshold (0-1)`. Absolute value that the target bin is scaled by and then compared with its neighbours.
- d. Example: A value of 0.5 means that the $0.5 * \text{the bin}$ must be greater than the neighbouring bins to qualify as howling.

4. Interframe peak magnitude persistence (IFPR)

- a. Compares the power of potential howling bins with their power history.
- b. Parameter: `IFPR Decay Multiplier`. The frame history is multiplied by this value and compared to the current power to qualify howling.
- c. Example: A value of 0.95 means the current frame bin must be within 95% of the power of the previous frame to qualify as howling.

5. Counter

- a. If a bin makes it through all four stages then a counter is incremented. The counter step size depends on the bin signal power: the louder the signal, the larger the step size to trigger the howling flag more quickly.
- b. Parameter: `Counter Trigger` determines the counter value that will generate a howling detection flag.
- c. Parameters `STEP_SIZE_nDB_FS` control how much the counter is incremented for each frame that a given bin has howling detected based on the magnitude of the signal (greater than ndB).

There are three bins that do not follow the precise sequence above:

- DC: The DC bin has a lot of variation so howling detection is not performed on this bin.
- Bin 1: A separate counter is used to count the number of detections in a given window because howling can be present but intermittent between frames. These are `BIN1 Trigger Detect Count` (how many frames will trigger howling detection) and `BIN1 Trigger Reset Count` (how many frames the counter is reset if no howling has been detected).
- Nyquist: The algorithm does not operate above the Nyquist frequency. Howling in these frequencies must be removed with a low-pass characteristic, either in the ANC filter or the ANC Hardware low pass filter.

Mitigating howling

When howling is detected, the gain is rapidly reduced to stop the howling. It is expected that howling is caused during a transient event, so the algorithm tries and recover the gain to the normal operating level. The following sequence of events occur:

1. Howling is detected.
2. The gain is reduced by a predetermined amount. This depends on whether howling was detected for the first time or during gain recovery.
 - a. When the howling is detected for the first time, the `Gain Decrement Step Size` parameter controls the gain decrement.
 - b. When the `Gain Decrement Step Size` parameter is applied, the gain continues to reduce for as long as howling is detected, until the gain reaches the value specified for **Minimum Gain Value** parameter .
 - c. If howling is detected during recovery, the `Successive Gain Decrement Step Size` parameter controls the gain decrement.
3. Recovery starts. After a certain number of frames, the gain is increased.
 - a. The `Frame Duration` parameter controls the number of frames between gain increments.
 - b. If recovering after howling is detected during recovery, the `Slow Frame Duration` parameter controls the number of frames between gain increments.
 - c. The `Gain Recovery Step Size` parameter controls the step size for this gain increment.

4. The recovery ends if it reaches the target gain. Otherwise, the cycle continues until the condition disappears.
5. If the gain mitigation is extreme, then slower recovery is applied.
 - a. The Threshold for Slowest Frame Duration parameter determines the threshold for slow recovery.
 - b. The Slowest Frame Duration parameter controls the number of frames between gain increments.

Figure 8-20 displays the gain recovery behavior.

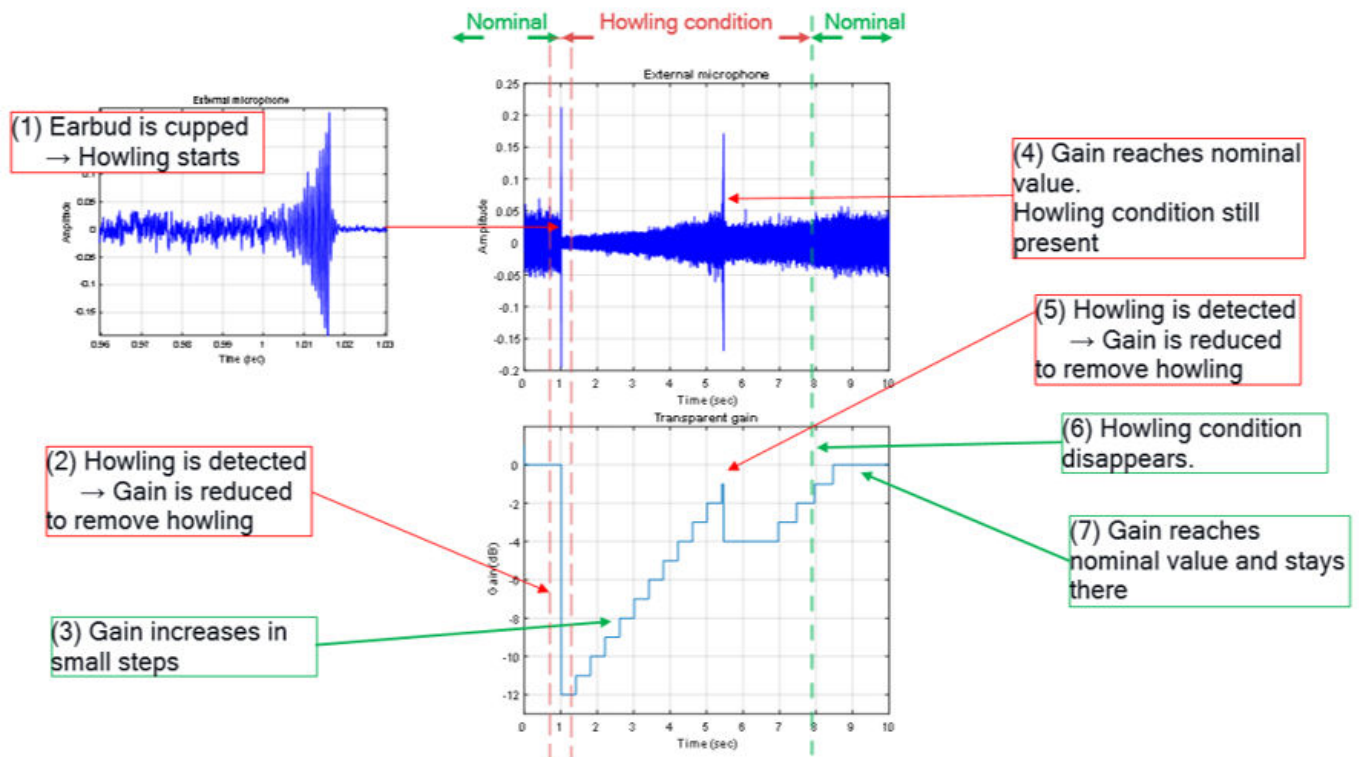


Figure 8-20 HCGR gain recovery

9 Static ANC tuning using ANC Filter Designer

Static ANC tuning involves designing feed-forward, feedback, echo canceller, and leak-through ANC IIR filters. These filters are static, meaning that they are not updated or changed after they are saved to the ANC file system. The ANC Filter Designer is a tool for designing ANC filters and performing static tuning in the ANC hardware.

9.1 ANC Tuning mode vs Mission mode

The Earbud Application and Headset Application provide two modes to perform static ANC tuning, Tuning mode and Mission mode.

Tuning mode

Tuning mode enables the user to make time-synchronized recordings to characterize the headset acoustic responses, such as passive attenuation and speaker response. Tuning mode requires a USB connection to collect the recordings. ANC parameters can also be configured in Tuning mode.

When the recordings are collected on a DUT with USB only connection in Tuning mode, the models can only be generated with feedback (error) microphone as reference. When using a different microphone to measure ANC performance, such as a HATS/ eardrum microphone, performance estimated by the ANC Filter Designer may not be an accurate representation of measured performance.

Mission mode

Mission mode has the same functionality of Tuning mode, except that it cannot perform USB recording. In Mission mode, the recordings are captured with both a feedback and HATS microphone as reference. When a headset has only a USB connection to the device, QUIL recommends using Mission mode .

For Automatic mode to work successfully, or for the ANC Filter Designer to estimate ANC performance accurately in Filter PEQ, generate the reference models by loading the recordings with an error (feedback) microphone, and a Drum (HATS) microphone as reference.

For information about the modes that support ANC filter design for each path, see [ANC filter design modes](#).

9.2 Launch the ANC Filter Designer

Before filter designing and tuning the ANC hardware, launch the ANC Filter Designer.

To launch the ANC Filter Designer:

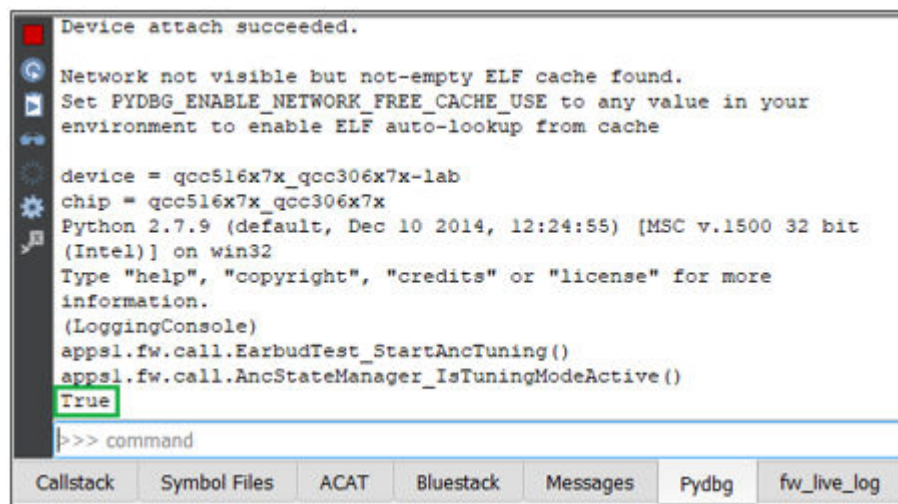
1. Navigate to {**chip_chipcode**} > **audio** > **bin** > **anc_filter_designer**.
2. Run `ancdesigner.exe`.

9.3 Connect to the DUT in Tuning mode

Connecting ANC to the DUT in Tuning mode requires three main steps: Enable Tuning mode using a Pydbg command, access configurations in the ANC Filter Designer, and connect to the device.

To connect ANC to the DUT in Tuning mode:

1. Run the following command in Pydbg to enable ANC in Tuning mode:
`apps1.fw.call.EarbudTest_StartAncTuning()`
2. Confirm that ANC is enabled in Tuning mode using the following command:
`apps1.fw.call.AncStateManager_IsTuningModeActive()`



```
Device attach succeeded.
Network not visible but not-empty ELF cache found.
Set PYDBG_ENABLE_NETWORK_FREE_CACHE_USE to any value in your
environment to enable ELF auto-lookup from cache

device = qcc516x7x_qcc306x7x-lab
chip = qcc516x7x_qcc306x7x
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
(LoggingConsole)
apps1.fw.call.EarbudTest_StartAncTuning()
apps1.fw.call.AncStateManager_IsTuningModeActive()
True
>>> command
```

Figure 9-1 ANC connected to the DUT in Tuning mode

3. Access configurations in the ANC Filter Designer:
 - a. Go to **File > Configuration > General**. Select the device details, and click **OK**.

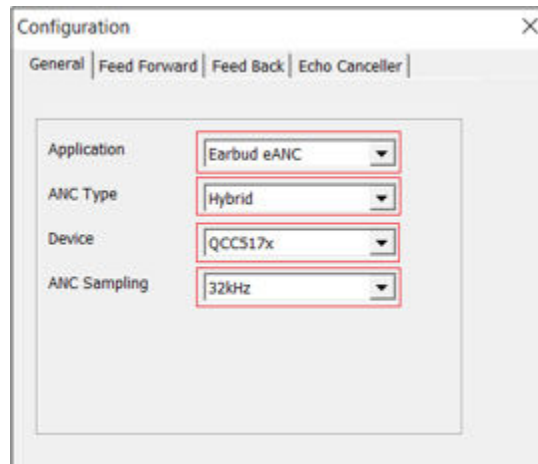


Figure 9-2 General tab in ANC Configuration dialog

- b. Go to **Device > Device Configuration**. Enter the device's transport method and the device ID.
 - c. Select **Access Device in Tuning Mode**.
 - d. In the **ADK toolkit location of** box, navigate to the device location and click **OK**.

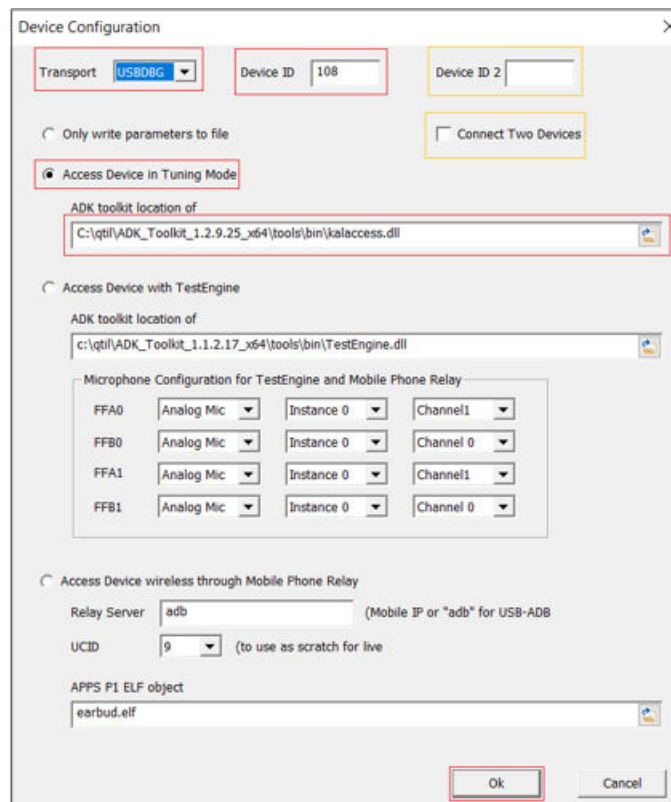


Figure 9-3 ANC Device Configuration in Tuning mode

4. Go to **Connect / Write**. When the ANC Filter Designer connects to the device, the text next to the Connect / Write box displays *Connected*.

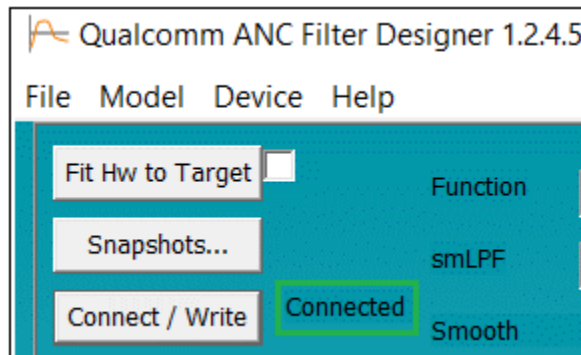


Figure 9-4 ANC Filter Designer connected to the device in Tuning mode

NOTE To enable more than one device, enter the device ID in the **Device Is 2** box and select the **Connect Two Devices** checkbox.

Table 9-1 lists Pydbg commands that can be used when tuning ANC in Tuning mode.

Table 9-1 Pydbg commands used in Tuning mode

Command	Description
<code>apps1.fw.call.EarbudTest_SetAncEnable()</code>	Enable ANC.
<code>apps1.fw.call.EarbudTest_SetAncDisable()</code>	Disable ANC.
<code>apps1.fw.call.EarbudTest_GetAncState()</code>	Get ANC State.
<code>apps1.fw.call.EarbudTest_SetAncMode(1)</code>	Set ANC mode 1.
<code>apps1.fw.call.appTestPhyStateOutOfCaseEvent()</code>	Take earbud out of case.
<code>apps1.fw.call.appTestPhyStateInCaseEvent()</code>	Put earbud inside the case.
<code>apps1.fw.call.EarbudTest_StartAncTuning()</code>	Put earbud in ANC tuning mode.
<code>apps1.fw.call.AncStateManager_IsTuningModeActive()</code>	Check if the ANC tuning mode is enabled.

NOTE To enable ANC in Run mode, the earbuds must be out of case.

9.4 Connect to the DUT in Mission mode

Connecting ANC to the DUT in Mission mode requires three main steps: Enable Mission mode using Pydbg commands, access configurations in the ANC Filter Designer, and connect to the device.

To connect ANC to the DUT in Mission mode:

1. Run the following commands in Pydbg to enable ANC in Mission mode:

```
apps1.fw.call.appTestPhyStateOutOfCaseEvent()
apps1.fw.call.EarbudTest_SetAncEnable()
```

2. Confirm that ANC is enabled in Mission mode using this command:

```
apps1.fw.call.EarbudTest_GetAncstate()
```


3. Access configurations in the ANC Filter Designer:
 - a. Go to **File > Configuration > General**. Select the device details, and click **OK**. The configuration must be the same as the Earbud Application built.

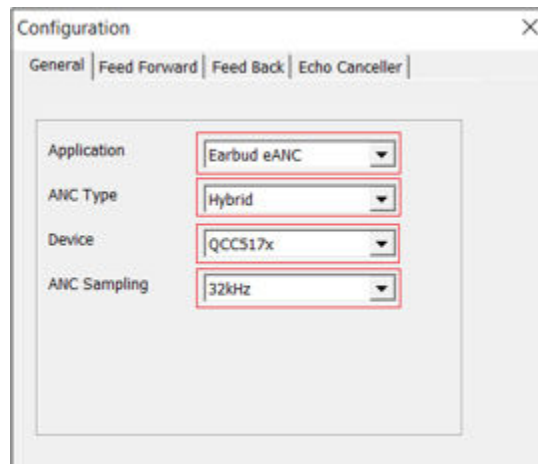


Figure 9-5 General tab in ANC Configuration dialog

- b. Go to **Device > Device Configuration**. Enter the device's transport method and the device IDs.
 - c. Select **Access Device with TestEngine**.
 - d. In the **ADK toolkit location of** box, navigate to the device location and click **OK**.

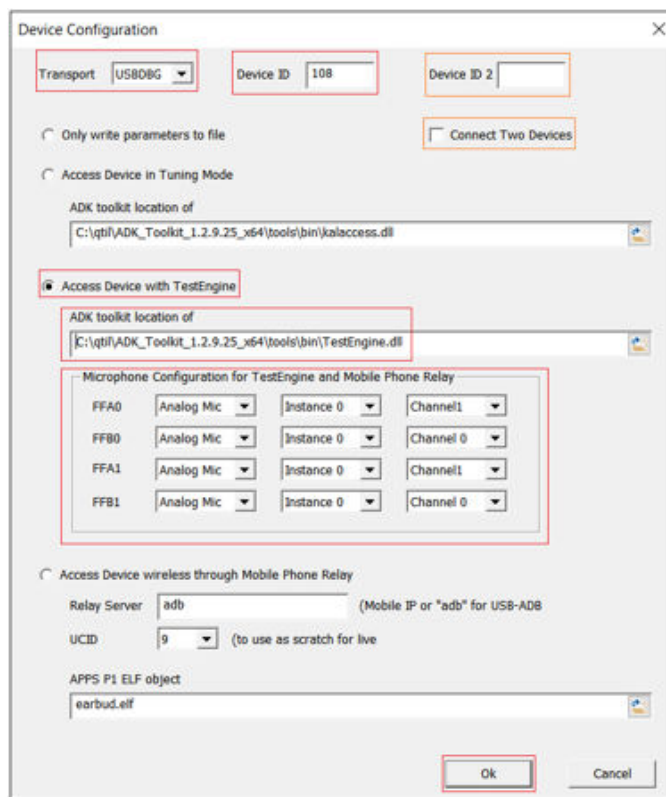


Figure 9-6 ANC Device Configuration Dialog in Mission mode

- Go to **Connect / Write**. When the ANC Filter Designer connects to the device, the text next to the Connect / Write box displays *Connected*.

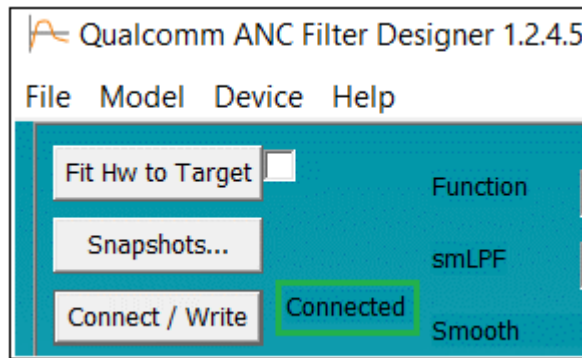


Figure 9-7 ANC Filter Designer connected to the device in Mission mode

Table 9-2 lists Pydbg commands that are used when tuning ANC in Mission mode.

Table 9-2 Pydbg commands used in Mission mode

Command	Description
<code>apps1.fw.call.EarbudTest_SetAncEnable()</code>	Enable ANC.
<code>apps1.fw.call.EarbudTest_SetAncDisable()</code>	Disable ANC.
<code>apps1.fw.call.EarbudTest_GetAncState()</code>	Get ANC State.
<code>apps1.fw.call.EarbudTest_SetAncMode(1)</code>	Set ANC mode 1.
<code>apps1.fw.call.appTestPhyStateOutOfCaseEvent()</code>	Take earbud out of case.
<code>apps1.fw.call.appTestPhyStateInCaseEvent()</code>	Put earbud inside the case.
<code>apps1.fw.call.EarbudTest_StartAncTuning()</code>	Put earbud in ANC tuning mode.
<code>apps1.fw.call.AncStateManager_IsTuningModeActive()</code>	Check if the ANC tuning mode is enabled.

9.5 ANC Filter Designer configuration

The ANC Filter Designer enables the configuration of ANC parameters for each of the feed-forward, feedback, and echo canceller paths. The General Configuration dialog includes configurations for device settings and filter optimization.

To open the Configuration dialog in the ANC Filter Designer, go to **File > Configuration**. Figure 9-8 shows how to select the Configuration dialog from the File menu.

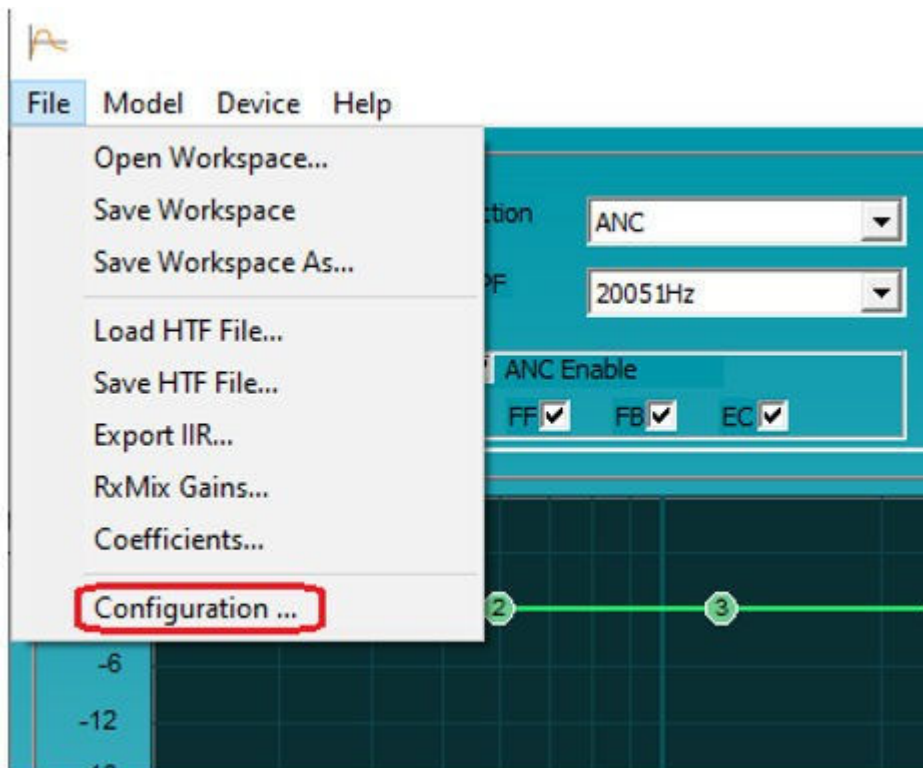


Figure 9-8 Opening the Configuration dialog in the ANC Filter Designer

General configuration

From **General** tab on the Configuration dialog, enter the following information:

- **Application:** Select between Earbud ANC, Earbud eANC, and Headset.
- **ANC Type:** Select between Hybrid, Feed-Forward, and Feed Back.
- **Device:** Select the device type.
- **ANC Sampling:** Select the clock frequency for the selected device. The recommended sampling is 32 kHz.

Figure 9-9 shows the Configuration dialog, with example General settings entered. For a full list of parameters, see [ANC Filter Designer mode parameters](#).

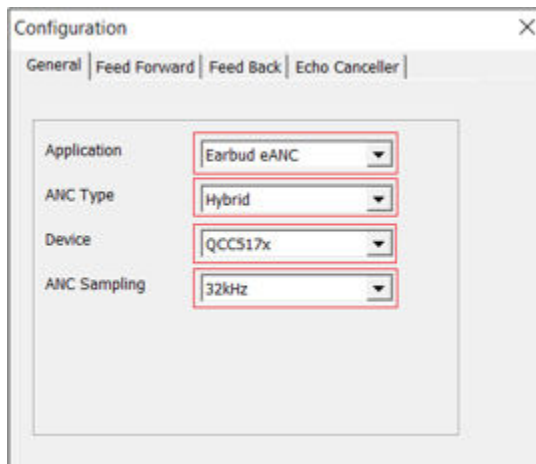


Figure 9-9 General configuration dialog

9.5.1 ANC filter design modes

ANC filter design for each path can be performed using three different modes: Automatic Design mode, Filter PEQ mode, and Automatic Hybrid mode (available in feed-forward ANC only). To select a mode in the ANC Filter Designer, go to **File > Configuration > (ANC path) > Filter design mode** on the Feed Forward/Feed Back tabs.

NOTE The *ANC path* can be feed-forward, feedback, or echo canceller

Automatic Design mode

In Automatic Design mode, the optimization algorithm finds an IIR filter that best matches a user-defined ANC performance.

The ANC Filter Designer takes the following inputs:

- Reference models
- ANC target performance
- Weighting to emphasize the frequency regions for the optimization algorithm to obtain the best possible ANC performance

Filter PEQ mode

Filter PEQ mode enables designing an IIR filter according to the frequency response specified by the PEQ model (target filter) in the ANC Filter Designer.

The optimization algorithm finds an IIR filter that best matches a defined filter target. The ANC Filter Designer takes the following inputs:

- Filter target response, designed with an optional reference model and a PEQ target filter
- Weighting to emphasize specific frequency regions for the fitting algorithm

Automatic Hybrid mode

Automatic Hybrid mode enables designing a feed-forward IIR filter based on the feedback ANC filter for a defined hybrid ANC performance

In general, it is recommended to use the Automatic Design mode. For feedback filter design, the ANC Filter Designer handles closed loop stability. For the Automatic Design mode to work efficiently, ensure that the reference models are accurate, see [Generate reference models in Tuning mode](#) and [Generate reference models in Tuning mode](#).

9.5.2 ANC Filter Designer mode parameters

The ANC Filter Designer mode parameters enable optimizing the filter design for each path.

Table 9-3 ANC Filter Designer mode parameters

Parameter name	Definition
Filter Design mode	Automatic: In Automatic Design mode, the optimization algorithm finds an ANC IIR filter that best matches a defined ANC and noise performance. The ANC Filter Designer takes the following inputs: - Model. - ANC target performance for low frequencies. - Required noise attenuation for high frequencies. - Additional weighting to emphasize specific frequency regions for the fitting algorithm. Filter: In Filter Design mode, the optimization algorithm finds an IIR filter that best matches a defined filter target. The ANC Filter Designer takes the following inputs: - Filter target response, designed with the optional reference model and a PEQ target filter. - Additional weighting to emphasize specific frequency regions for the fitting algorithm.
Num optimization frequencies	Number of discrete IIR frequency points used by the filter optimization algorithm. A lower value speeds up the algorithm but may be less precise. Especially sharp filter transitions, for example, peaks or notches, may be missed if the distance between the frequencies is too large.
Min optimization frequency	The optimization algorithm does not try to improve ANC performance below this frequency. Lower frequencies are still used for stability testing for the feedback path.
Max optimization frequency	The optimization algorithm does not try to improve ANC performance above this frequency. Instead, the algorithm tries to keep the noise amplification below 0 dB.
Noise Attenuation above Fmax (dB)	Keeps the filter amplification above maximum optimization frequency below the defined level.
Feedback Gain margin (dB) ^(a)	Defines minimum gain margin used by the filter optimization algorithm in ANC Filter Design mode.
Feedback Phase margin (degrees) ^(a)	Defines minimum phase margin used by the filter optimization algorithm in ANC Filter Design mode.
Feedback Max Open Loop gain (dB) ^(a)	Maximum open loop gain allowed while generating the feedback ANC IIR filter.

Table 9-3 ANC Filter Designer mode parameters (cont.)

Parameter name	Definition
Low frequency Open Loop gain limit (Hz) ^(a)	The lower bound of frequency at which the ANC IIR filter need to be optimized for a given model.
Minimum gain (dB)	Defines the minimum gain that can be applied by the optimization algorithm while generating the ANC IIR filter. Use 0 dB.
Maximum gain (dB)	Defines the maximum gain that can be applied by the optimization algorithm while generating the ANC IIR filter.

^a Feedback ANC only

9.6 Access device with Wireless Debug

Instead of using Pydbg, the ANC Filter Designer can handle the connection internally using a Wireless Debug connection.

For the initial steps to setup for Wireless Debug, see the *End to End Wireless Debug User Guide*.

To avoid conflicts during access, limit use of the Wireless Debug connection to one client at a time, either the ANC Filter Designer or Pydbg.

The specific settings in the Wireless Debug dialog are:

- The default value for Relay Server is *adb*, which uses the Android Debug Bridge to connect to the mobile phone relay. This enables the app to automatically start on the mobile phone.
- When the IP address of the mobile phone is supplied instead, USB ADB connection to the phone is not required, but there are some limitations. For example, the app cannot be started automatically.
- The supplied UCID is used as scratch to transfer the ANC settings. It is overwritten with temporary settings during tuning connection. To avoid interference with the normal application ANC modes, the default is to use the last entry (9) at the end of the UCID range.
- The APPS P1 ELF object file is required to be supplied to get information on code/data locations. During the connection sequence, the build version information in the ELF file is checked to match the version running on the device.

NOTE To work with an earbud pair, check the **Connect Two Devices** check box.

- The Wireless Debug connection is significantly slower, compared to other methods. Besides the limited speed due to Bluetooth transmission, it can also vary depending on the other components involved, such as a PC, network, or mobile phone.
- Typical speed is about 3 s to update a ANC configuration. The initial connection takes slightly longer, depending on whether the mobile phone app is already started and connected.

Wireless Debug test commands

If the device is connected by Wireless Debug, the connection can send a limited number of test commands to device.

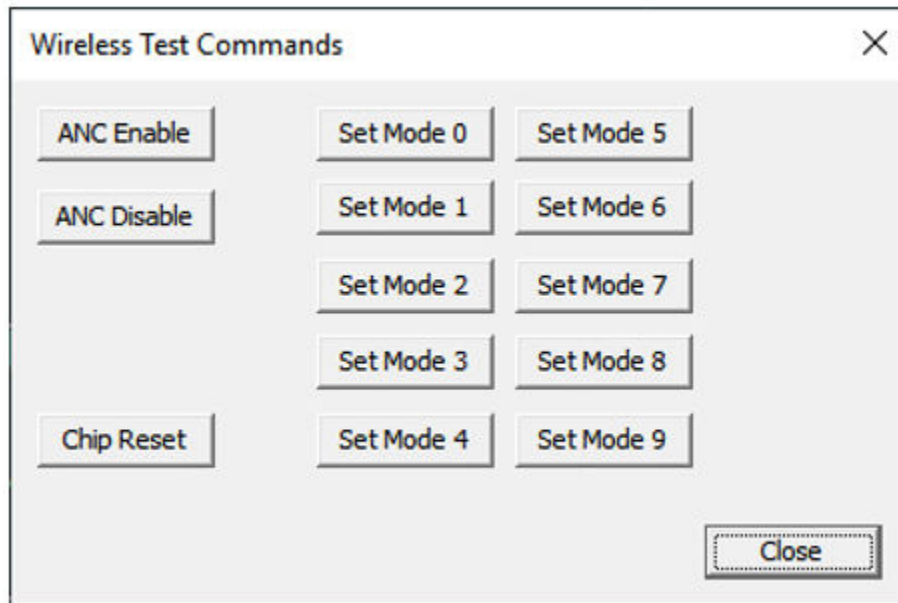


Figure 9-10 Wireless Debug Test Commands dialog

- ANC Enable/ANC Disable can switch ANC on/off.
- Set Mode 0...9 changes ANC mode to a given UCID.
- Chip Reset can reset the device in some cases, but is not required during normal tuning.

NOTE Issued commands might interfere with operation in other areas of the ANC Filter Designer. For example, when ANC is switched off, it is not automatically switched back on when writing new ANC settings to a device.

9.7 Generate acoustic path models

Models of the acoustic paths are needed for the ANC Filter Designer to tune ANC filters and estimate ANC performance accurately.

For the ANC Filter Designer to use these acoustic path models, use either Tuning mode or Mission mode to tune filters for models and generate reference models.

9.7.1 Collect recordings in Tuning mode

Various models of the acoustic paths are needed to tune ANC filters and to estimate ANC performance accurately in the ANC Filter Designer. To generate these models, collect and load the recordings into the ANC Filter Designer.

Before collecting recordings in Tuning mode, ensure that the ANC Tuning mode is enabled as described in [Connect to the DUT in Tuning mode](#).

Tuning mode chamber setup

For best results, collect ANC recordings and performance measurement in Tuning mode using a chamber of multi-directional source sources. This setup helps to generate filters that perform with noise coming from various directions.

Figure 9-11 shows a test chamber in Tuning mode, where the external speakers play white/pink stimulus noise into the chamber and the level measured at DUT should be at 85 dB SPL.

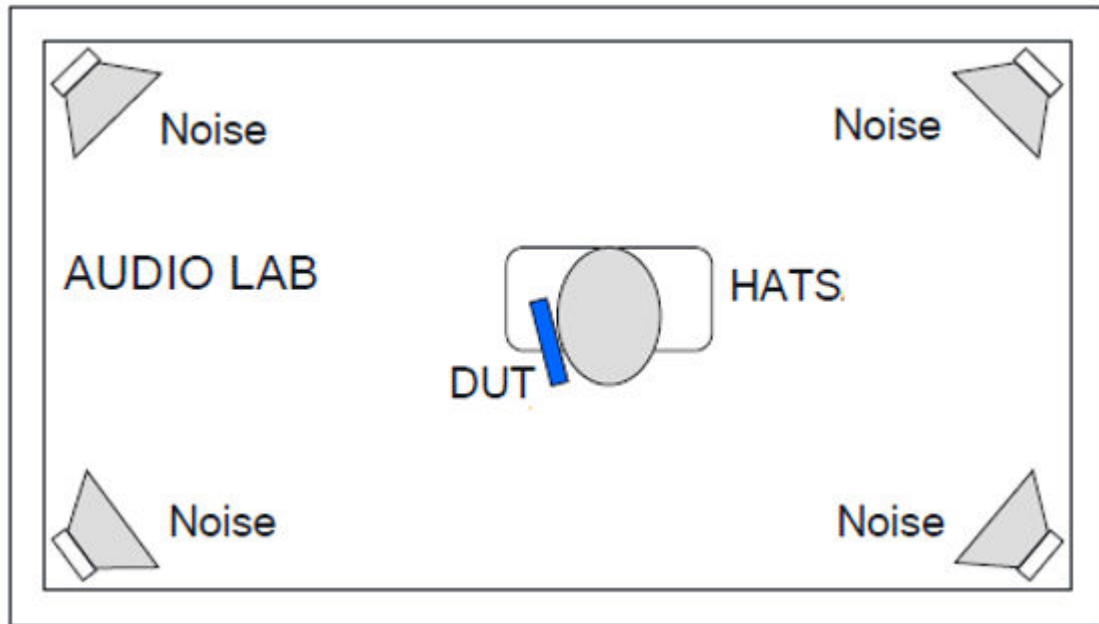


Figure 9-11 Tuning mode chamber setup

Collect S-path and E-path recordings

To collect S-path and E-path recordings in Tuning mode:

1. From the ANC Filter Designer, go to **Device > Tuning Mode > Path Recording > Recording Preset**.
2. In **Recording Preset**, select **S-path** or **E-path**, see Figure 9-12.
3. Use audio acquisition software, for example, an Audacity session:
 - a. Import a stimulus noise, such as white or pink noise.
 - b. Map the microphone and speaker to a QCC517x or later device.
 - c. Add a new stereo track from **Tracks > Add new > Stereo Tracks**.
 - d. Click the **record** button.

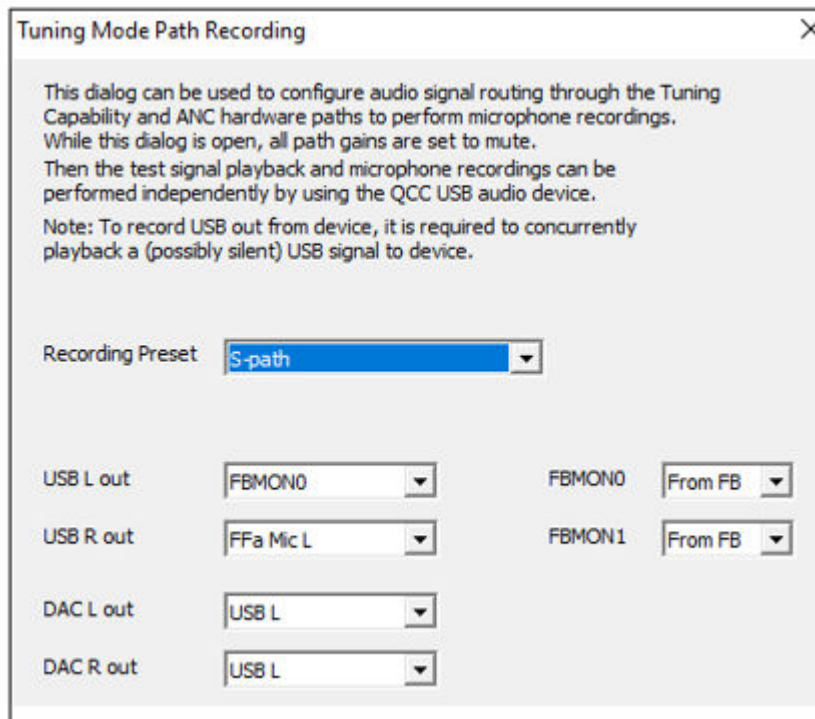


Figure 9-12 Selecting S-path in Tuning mode Path Recording dialog

NOTE Leave this window open while recording. This bypasses any filters loaded in an ANC path.

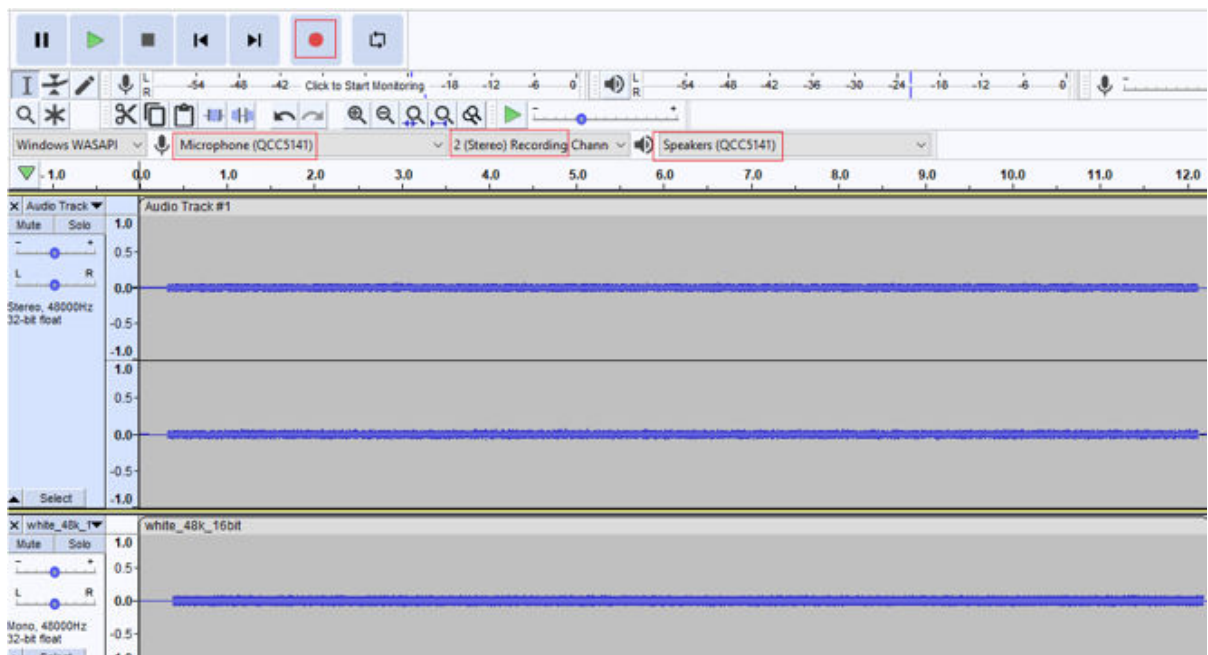


Figure 9-13 S-path recording

To save the recording when it is complete:

1. Select the recorded track.
2. Go to **File > Export > Export selected audio**.

Collect P-path recordings

To collect P-path recordings in Tuning mode:

1. From the ANC Filter Designer, go to **Device > Tuning Mode > Path Recording > Recording Preset**.
2. In **Recording Preset**, select **P-path Ext**, see [Figure 9-14](#).
3. Use an external speaker(s) to play white noise in the chamber (85 dB SPL is recommended).
4. No stimulus must be played through headset speaker during P-path recordings.

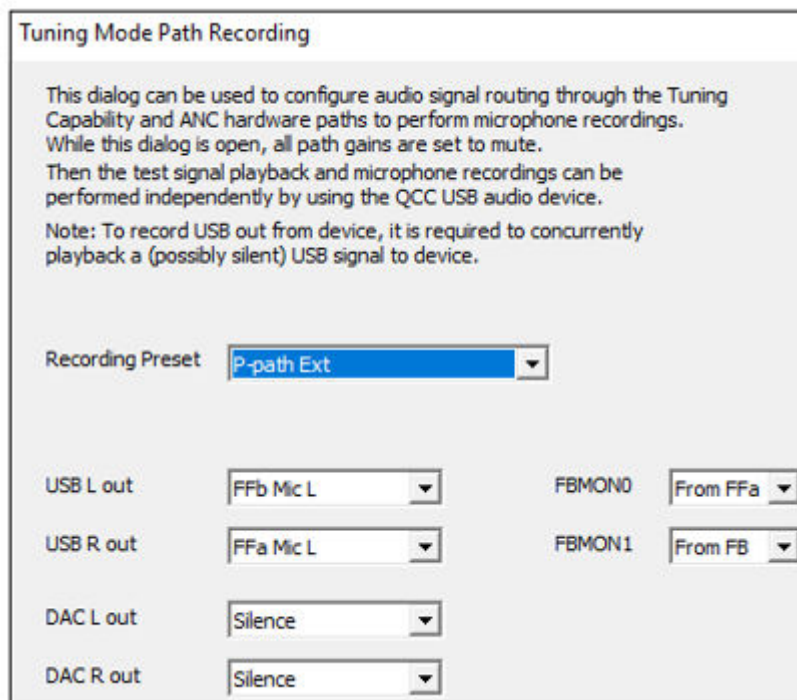


Figure 9-14 Selecting P-path Ext in Tuning Mode Path Recording dialog

NOTE Leave this window open while recording.

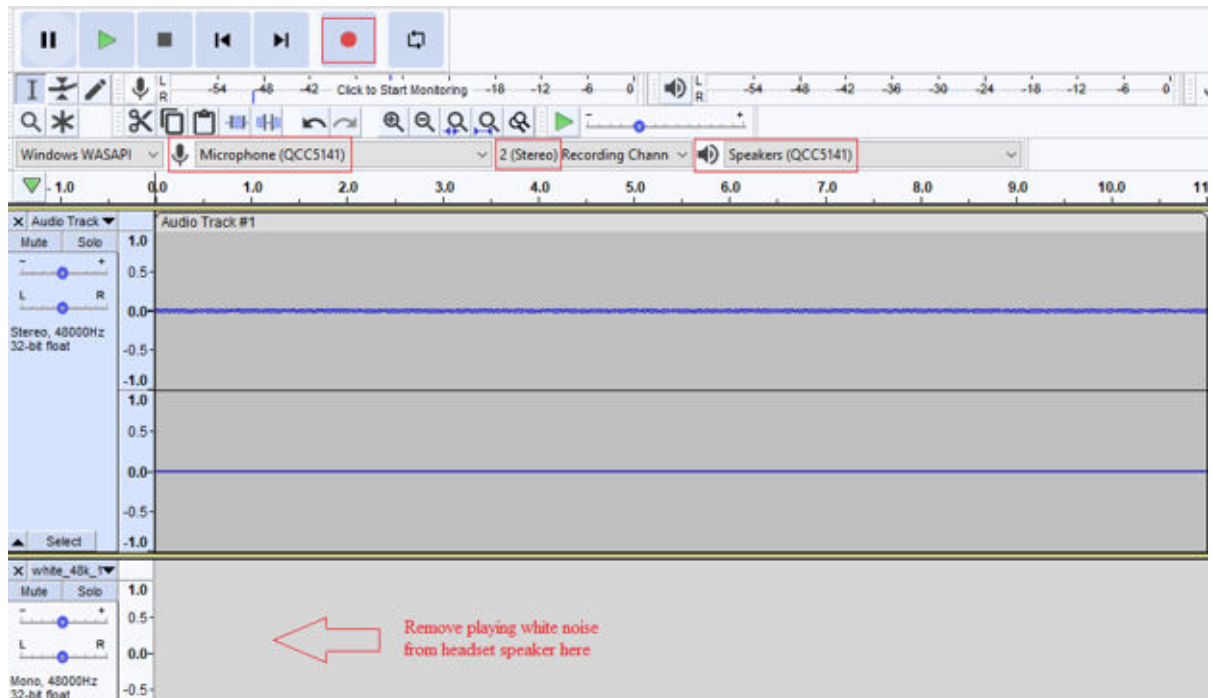


Figure 9-15 P-path recording

To save the recording when it is complete:

1. Select the recorded track.
2. Go to **File > Export > Export selected audio**.

Examples

Figure 9-16, Figure 9-17, and Figure 9-18 show example recordings in Tuning mode. In these examples:

- The P-path recording contains the feed-forward microphone and feedback microphone signal.
- The S-path recording contains the headset speaker signal and feedback microphone signal.
- The E-path recording contains the headset speaker signal and feed-forward microphone signal. The E-path recording is mainly used to check the quality of isolation between the headset speaker and external microphone.

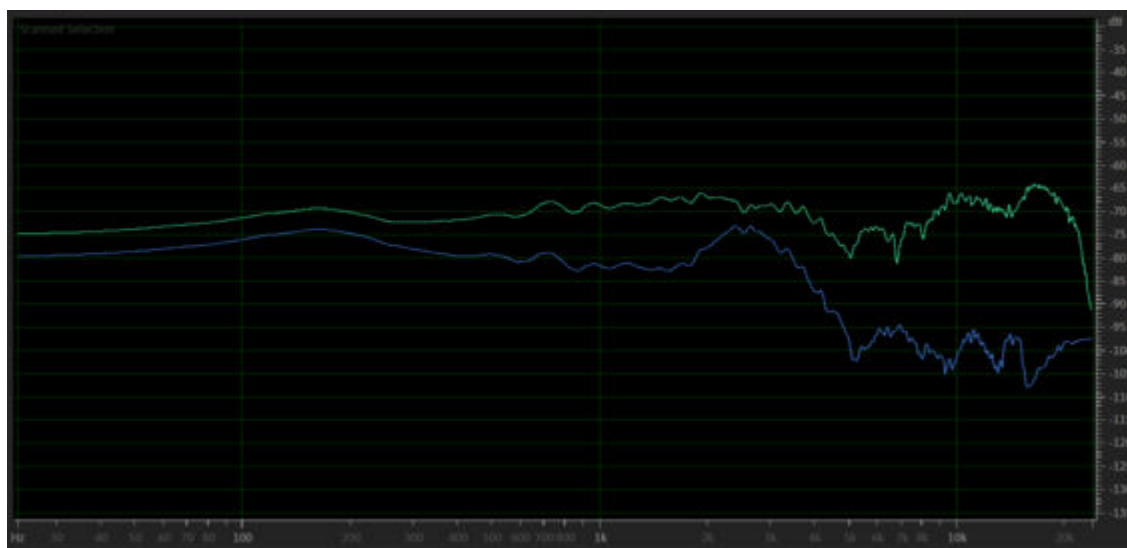


Figure 9-16 P-path recording in Tuning mode example

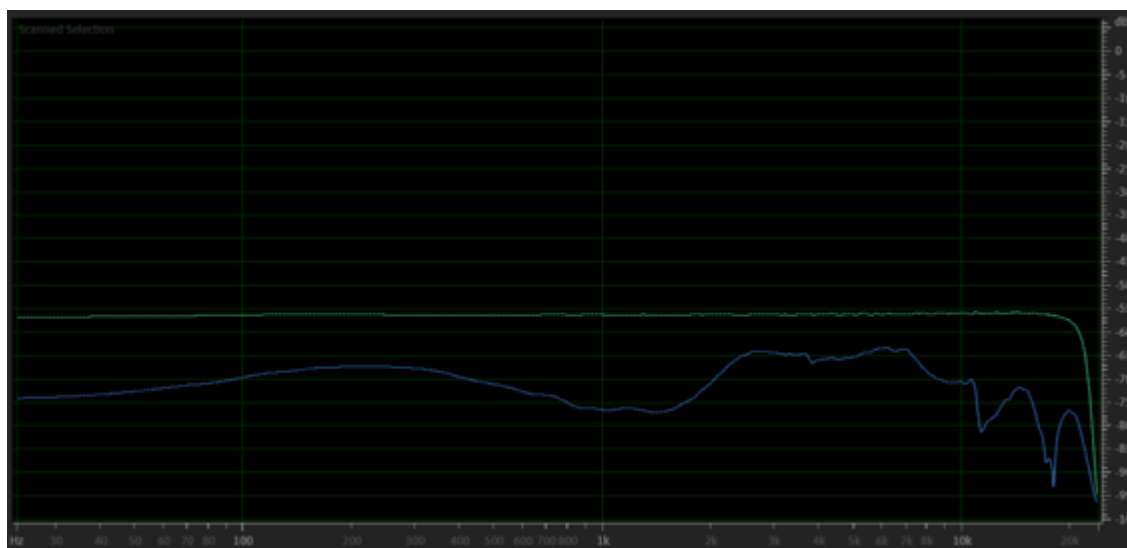


Figure 9-17 S-path recording in Tuning mode example

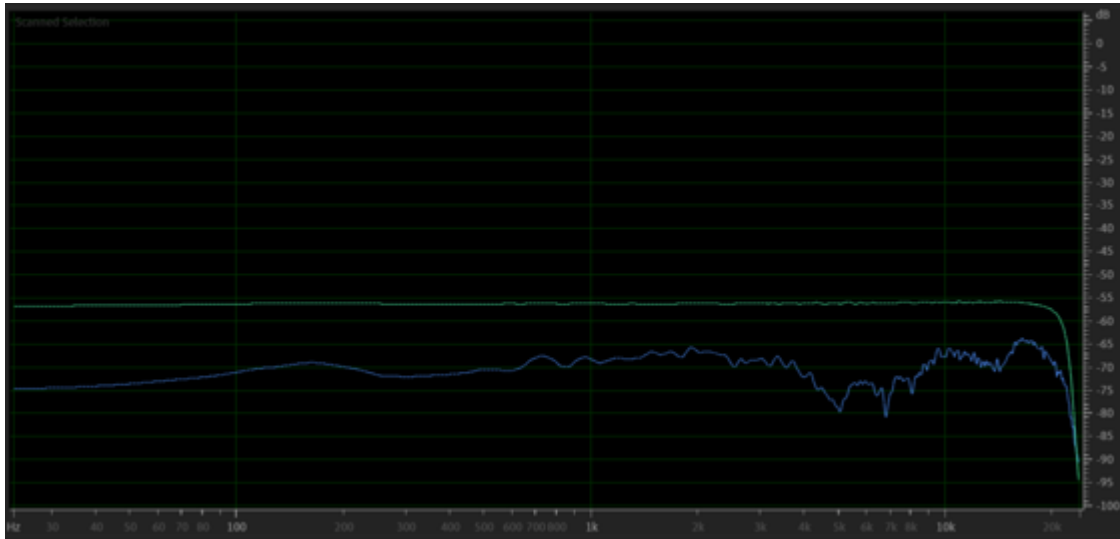


Figure 9-18 E-path recording in Tuning mode example

9.7.2 Collect recordings in Mission mode

Various models of the acoustic paths are needed to tune ANC filters and to estimate ANC performance accurately in the ANC Filter Designer. To generate these models in Mission mode, collect and load the recordings into the ANC Filter Designer.

Before collecting recordings in Mission mode, ensure that the ANC Mission mode is enabled as described in [Connect to the DUT in Mission mode](#).

Mission mode chamber setup

Mission mode chamber setup requires a hardware setup to take stereo recordings that contain an eardrum microphone signal and external loudspeaker signal.

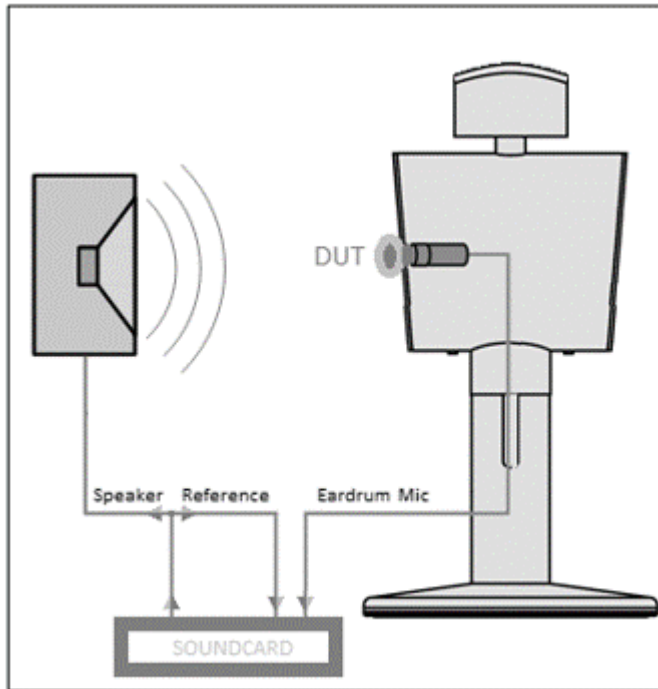


Figure 9-19 Mission mode chamber setup

The Mission mode uses directional noise setup. Only one speaker should be used with a multi-tone/ sine sweep stimulus file. QTIL recommends that the ambient noise should be at 85 dB SPL at the eardrum.

To set up the chamber in Mission mode to collect recordings, open an audio acquisition software, such as an Audacity session, and complete the following tasks:

1. Import a multi-tone stimulus from **map > map the mic to HATs eardrum and speaker to external Loudspeaker**.
2. Create a new stereo track from **Tracks > Add new > Stereo Tracks**.
3. Click the **record** button. It is required to take these recordings for at least 10 s.

To save the recording when it is complete:

1. Select the recorded track.
2. Go to **File > Export > Export selected audio**.

Example

Figure 9-20 shows an example recording collected in Mission mode.

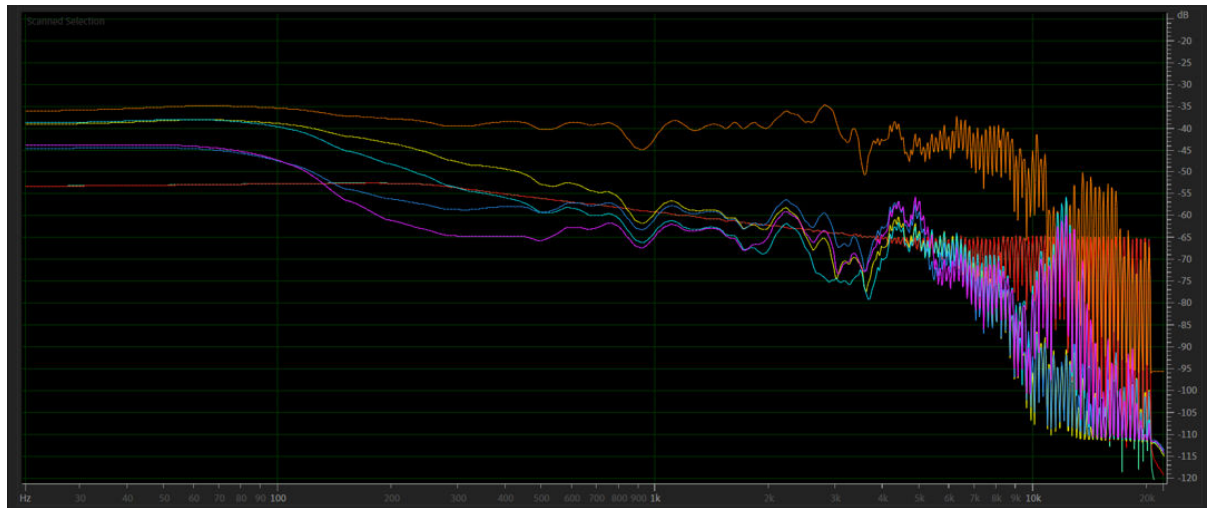


Figure 9-20 Example recording in Mission mode

- NOTE** The colored lines in Figure 9-20 signify:
- Red: Multitone from external speaker
 - Orange: Pinna
 - Yellow: PNC
 - Cyan: Feed-forward ANC
 - Blue: Feedback ANC
 - Pink: Hybrid ANC

9.7.3 Tune filters for models

Before taking a recording in Mission mode, it is necessary to have feed-forward and feedback IIR filters in Mission mode that have some level of ANC performance with the DUT. This aids in model generation

Pre-tune feed-forward to take a recording

To pre-tune feed-forward to take a recording:

1. Start with a flat filter with DC notch at 20 Hz and LPF at 5 kHz.
2. Adjust HW Gain (Gain Offset) and check Invert IIR until the ANC performance is below -6 dB @ 100..200 Hz. To improve performance, change the LPF and DC notch.
3. Take the recording.

- NOTE** Do not use a real feed-forward IIR filter for model generation, because the low pass characteristics causes inaccurate model estimation at high frequencies.

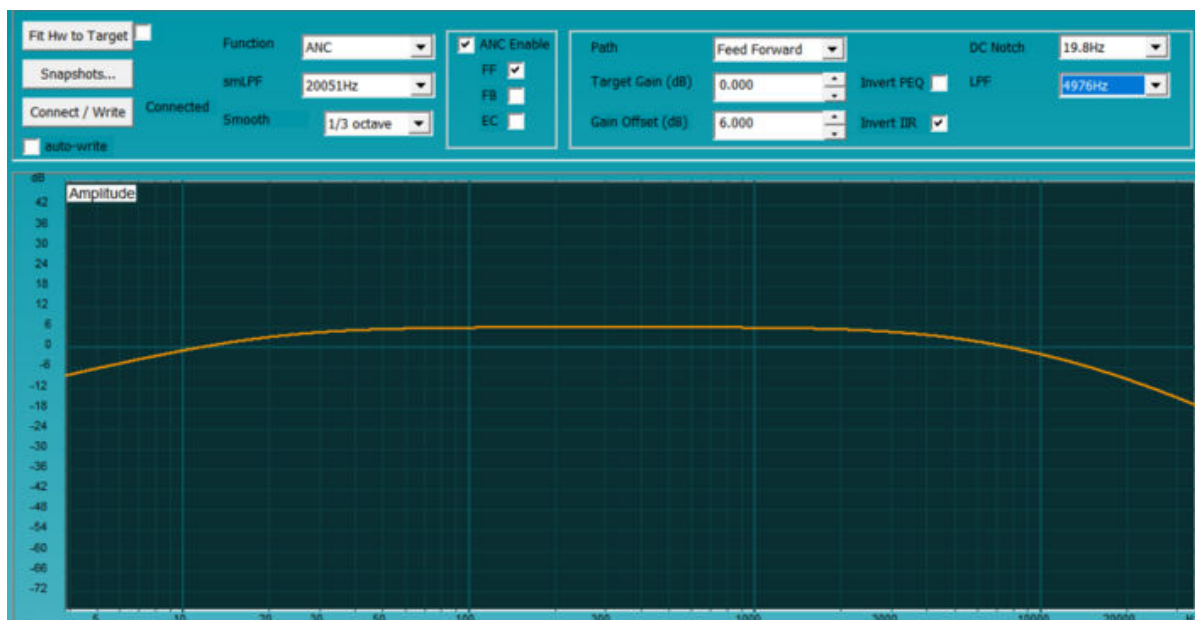


Figure 9-21 Pre-tune feed-forward recording in Mission mode

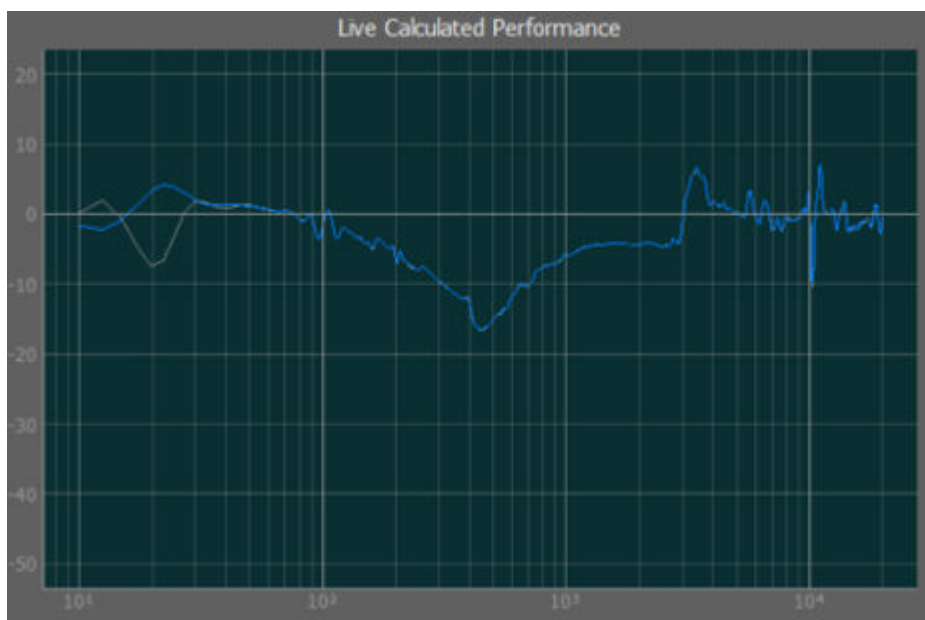


Figure 9-22 Feed-forward performance in Mission mode

Pre-tune feedback ANC to take a recording

To pre-tune feedback to take a recording:

1. Start with a flat filter with DC notch at 80 Hz and LPF at 1.2 kHz.
2. Set Gain Offset as large as possible, just below when the earbuds start to howl.
 - a. Increase the Gain Offset until howling starts.
 - b. Reduce the level by 1dB, or until there is no more howling.
 - c. Check if Invert IIR allows the Gain Offset to be increased until there is some ANC performance @ 100..200 Hz.
3. Take the recording.

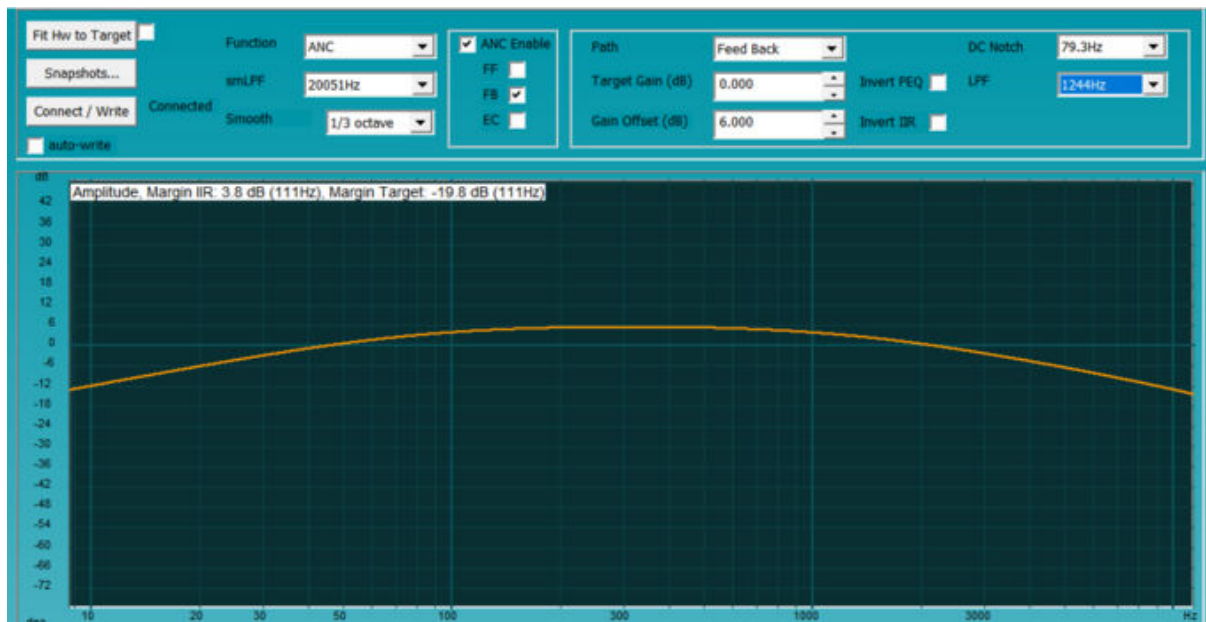


Figure 9-23 Pre-tune feedback recording in Mission mode

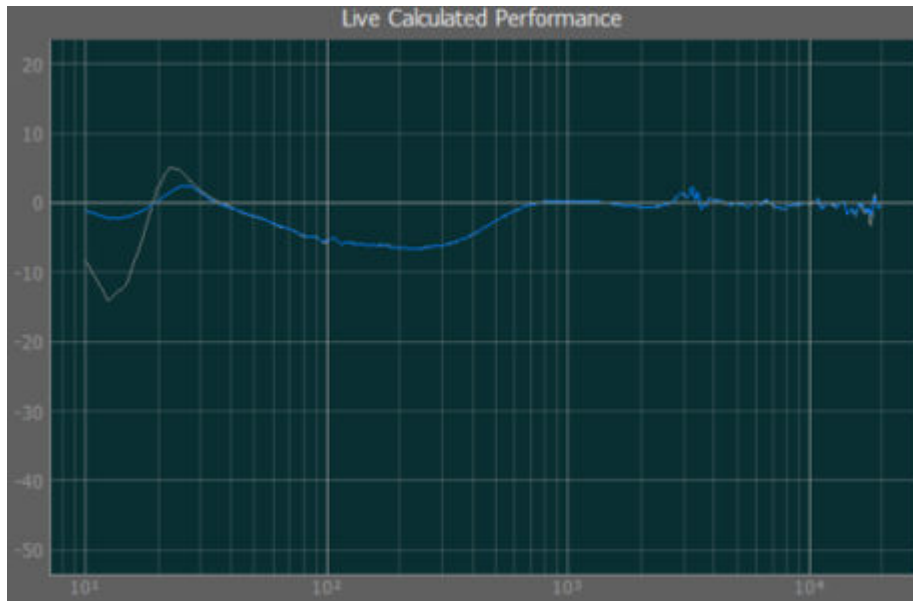


Figure 9-24 Feedback performance in Mission mode

9.7.4 Generate reference models in Tuning mode

Reference models in can be generated in Tuning mode from the S-path and P-path measurement .wav files.

The Model menu in the ANC Filter Designer provides methods to generate, import, export, and remove reference models

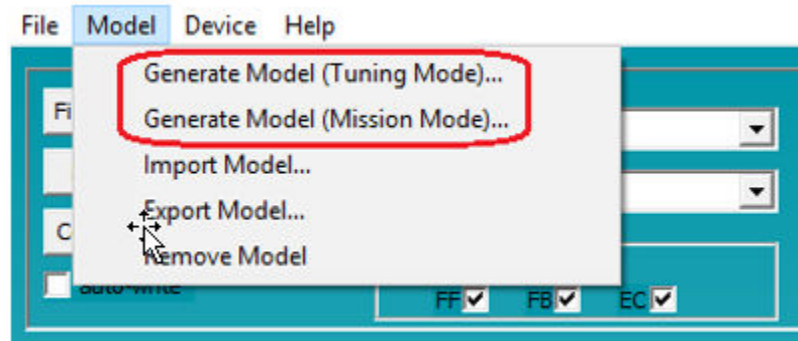


Figure 9-25 Generate a reference model in Tuning mode menu

To generate a reference model in Tuning mode:

1. From the ANC Filter Designer, go to **Model > Generate Model**.
2. Input the PD path, SD path or PE path, SE path, or all four paths.
3. Click **Generate**.

Figure 9-26 shows an example of model generation in Tuning mode from the PE path and SE path.

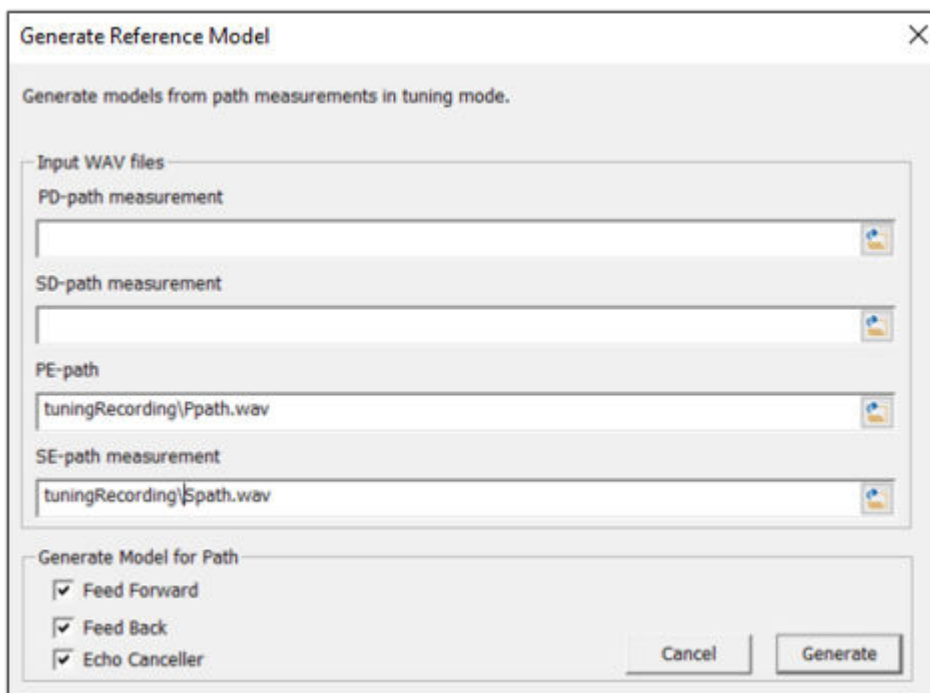


Figure 9-26 Generate a reference model in Tuning mode example

Figure 9-27 shows feed-forward as the path to tune.

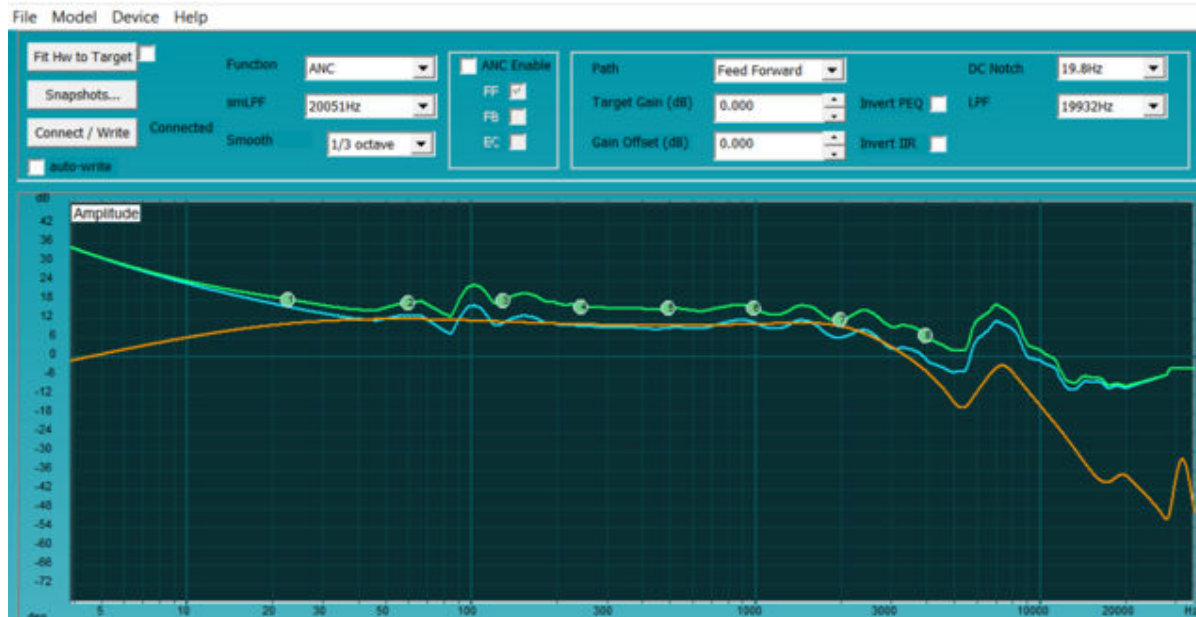


Figure 9-27 Tuning the feed-forward path in Tuning mode

NOTE The colored lines in Figure 9-27 signify:

- Blue: Reference model
- Orange: Real IIR hardware filter that can fit to the reference model
- Green: The PEQ reference for the fitting.

9.7.5 Generate reference models in Mission mode

Reference models are generated based on the recordings.

The Model menu in the ANC Filter Designer provides methods to generate, import, export, and remove reference models

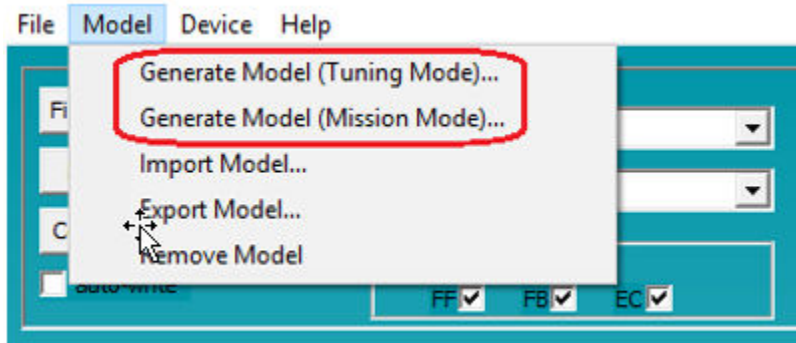


Figure 9-28 Generate a model in Mission mode menu

To generate models in Mission mode:

1. From the ANC Filter Designer, go to **Model > Generate Model**.
2. Input all five recording files:
 - Pinna
 - ANC OFF
 - ANC ON Feed Forward only
 - ANC ON Feed Back only
 - ANC ON Hybrid (Feed Forward + Feed)
3. Click **Generate**.

Figure 9-29 shows an example of model generation in Mission mode.

The 'Generate Model' dialog box is shown with the following settings:

- Test Signal:**
 - Signal type: Multitone
 - Reference on right channel: ☒
 - Frame: 16384
- Input WAV files:**
 - Pinna: missionRecording\pinna.wav
 - ANC OFF: missionRecording\PNC.wav
 - ANC ON Feed Forward only: missionRecording\FF.wav
 - ANC ON Feed Back only: missionRecording\FB.wav
 - ANC ON Hybrid (Feed Forward + Feed Back): missionRecording\HY.wav
- Generate Model for Path:**
 - Feed Forward: ☒
 - Feed Back: ☒
 - Echo Canceller: ☒

Buttons: Cancel, Generate

Figure 9-29 Generate a reference model in Mission mode example

Figure 9-30 shows feed-forward as the path to tune.



Figure 9-30 Tuning the feed-forward path in Mission mode

NOTE The colored lines in Figure 9-30 signify:

- Blue: Reference model.
- Orange: Real IIR hardware filter that can fit to the reference model.
- Green: The PEQ reference for the fitting.

9.7.6 Import, export, and remove reference models

The ANC Filter Designer provides options to import a saved model, export a model, or remove an existing model under the Model menu.

[Import, export, and remove reference models](#) shows the Model menu to import, export, ore remove a model.

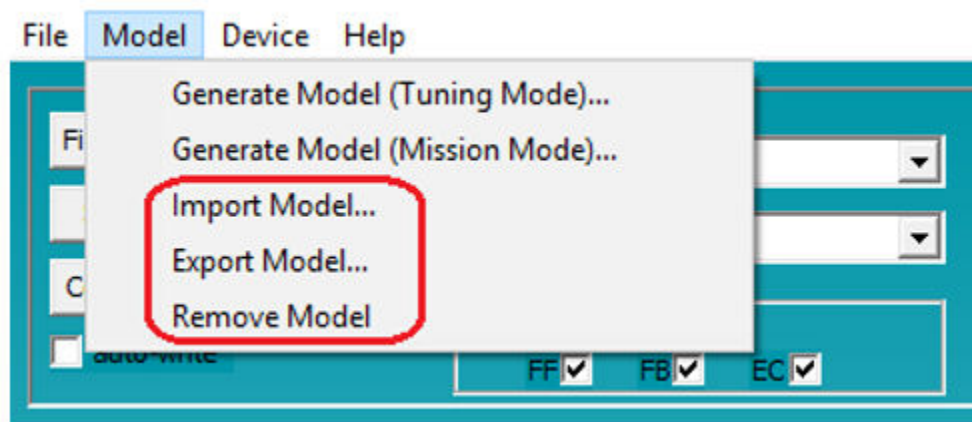


Figure 9-31 Import, export, and remove a model

The reference models are saved in *ypeq* or *csv* format. The model file contains frequency, amplitude, and phase information of path models in the following order, separated by a comma and a space:

Frequency, PDa, PDp, PEa, PEp, SDa, SDp, SEa, SEp, PD/SDa, PD/SDp

Table 9-4 Reference model file format

File	Description
Frequency	Frequency points as column vector.
PDa	PD path magnitude response at the frequencies specified in the frequency vector.
PDp	PD path phase at the frequencies specified in the frequency vector.
PEa	PD path magnitude response at the frequencies specified in the frequency vector.
PEp	PD path phase at the frequencies specified in the frequency vector.
SDa	SD path magnitude response at the frequencies specified in the frequency vector.
SDp	PD path phase at the frequencies specified in the frequency vector.
SEa	PD path magnitude response at the frequencies specified in the frequency vector.
SEp	PD path phase at the frequencies specified in the frequency vector.

9.7.7 Load a model generated in other tools

To load an externally generated model (for example, using Matlab) to the ANC Filter Designer, load a *ypeq* file containing the model's information to the ANC Filter Designer to use as a reference model.

NOTE It is not mandatory to include all the paths. The ANC Filter Designer loads the paths that exist in the *ypeq* file and ignores paths that do not exist.

9.8 Modify PEQ bands

PEQ bands can be modified for Target frequency response and weighting curves.

To modify PEQ bands, either:

- Drag the selected marker to change center frequency and gain. Change the quality with the mouse wheel.
- Open a dialog window by double-clicking on the marker. Change PEQ band parameters, including the PEQ type, as required.

9.9 Feedback ANC filter design

To design a feedback ANC filter, configure feedback ANC, then perform automatic tuning and advanced tuning.

The pre-requisites to tuning feedback ANC are:

- Configuration settings, see [ANC Filter Designer configuration](#)
- Generate/load reference models (if using Automatic Filter Design mode), see [Generate reference models in Tuning mode](#) or [Generate reference models in Mission mode](#)

9.9.1 Configure feedback ANC

The Configuration dialog for the feedback path has additional parameters to define the minimum feedback gain margin and minimum feedback phase margin used by the filter optimization algorithm.

To adjust the configuration for the Feed-back path, select the **Feed Back** tab on the Configuration dialog.

To configure the Feedback Gain Margin and Feedback Phase Margin, define the minimum gain and phase margin used by the filter optimization algorithm.

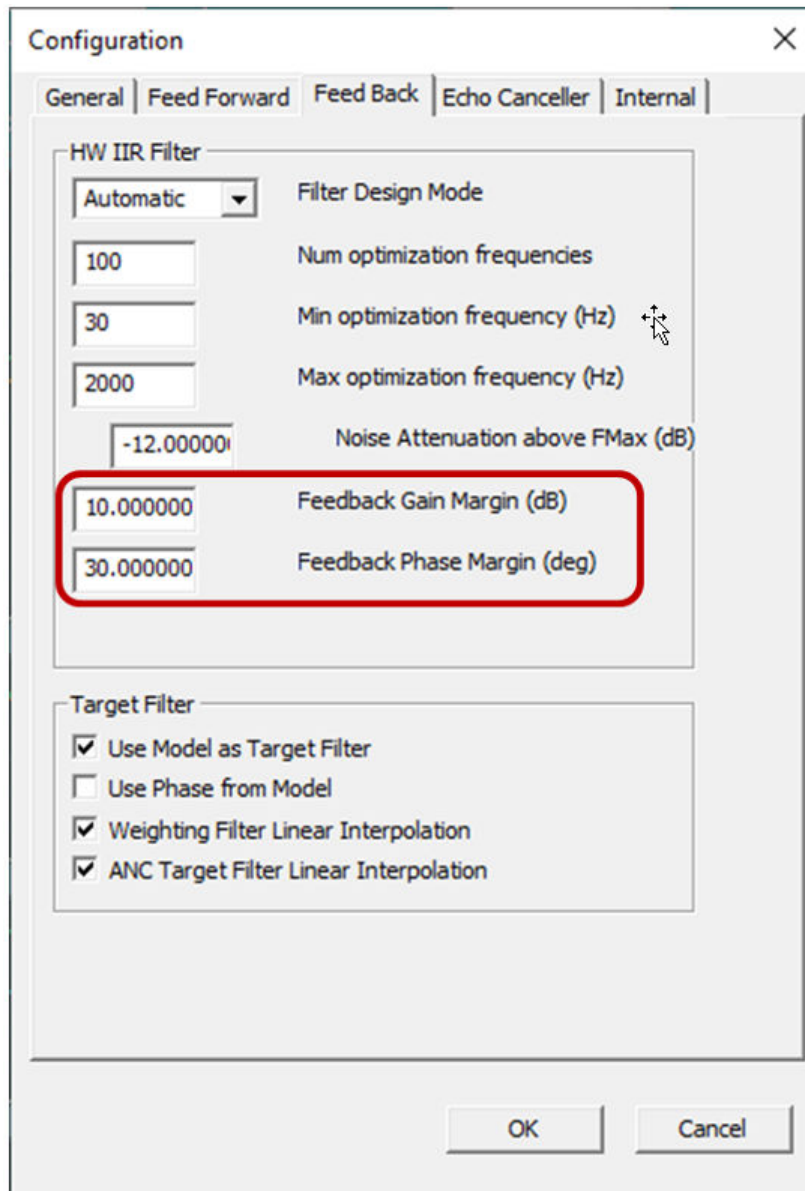


Figure 9-32 Adjusting Feedback Gain Margin and Feedback Phase Margin in Configuration dialog

9.9.2 Automatic feedback tuning

Automatic feedback tuning involves generating the feedback ANC filter using Automatic mode.

To set Automatic mode, see [ANC filter design modes](#).

To generate the feedback ANC filter using Automatic mode:

NOTE Each step has a corresponding white number in [Automatic feedback tuning](#).

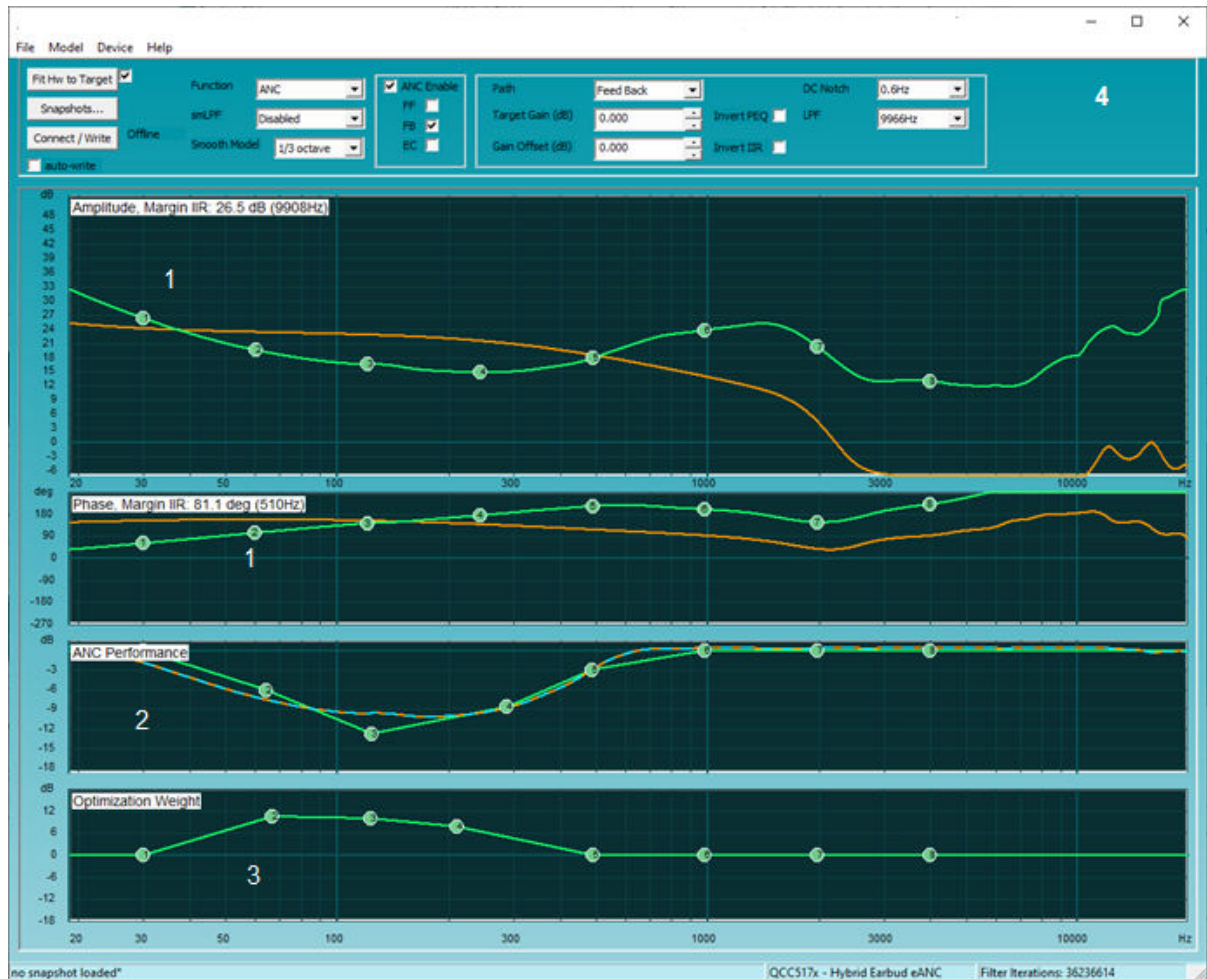
1. Generate/load the SE/SD path models of the DUT using the Model menu.
2. Use the PEQ curve in the ANC performance plot to set the target ANC performance.

NOTE The PEQ curve is only activated if the automatic mode is selected in the feedback ANC configuration settings.

3. Use the PEQ curve in the optimization weight to emphasize the frequencies for the fitting algorithm to find the IIR filter that closely matches the ANC performance in the frequencies with more weight.
4. Set the tuning parameters for the filter using the parameters in [Table 9-5](#).

Table 9-5 Feedback ANC parameters

Parameter	Description
Fit HW to target	Start filter optimization algorithm. It finds the IIR filter based on QCC517x ANC hardware that best matches the defined ANC performance (or based on the target amplitude and phase PEQ in Filter PEQ mode) NOTE A button press starts a limited number of optimization cycles. Checking the checkbox next to the button runs the fitting algorithm in a separate thread until the checkbox is un-checked.
Polarity	If the speaker polarity of DUT is inverted from the reference models loaded, the invert PEQ or invert IIR need to be selected. The definitions are: ■ Invert PEQ: Apply 180° phase shift to the target PEQ. ■ Invert IIR: Apply 180° phase shift to IIR filter that is written to the device.
Gain	To obtain the best ANC performance, apply the appropriate gain to the IIR filter. If additional gain is needed, apply it through Target gain or Gain offset.
Target gain	Gain of the target PEQ.
Gain offset	Gain offset that is applied to the ANC hardware without modifying the target filter.
LPF	The LPF dropdown can control the noise overshoot, out of band noise aliasing into audio band.



Automatic feedback tuning using Automatic mode

9.9.3 Advanced feedback tuning

Advanced feedback tuning refers to the fine tuning that is sometimes needed to optimize the ANC performance or to overcome the instability. This includes feedback filter PEQ tuning, controlling the noise overshoot, and using advanced plot options.

Some scenarios that require advanced feedback tuning include:

- Inaccurate reference models.
- Models are obtained with feedback (headset internal) microphones as reference, but the ANC performance is measured at another reference point, such as HATS.
- To suit the required frequency response.

Feedback Filter PEQ tuning

To define a unique filter shape, select the **Filter PEQ** option in the **Feedback Configuration** dialog and uncheck the **use model as target filter** box. Adjust the optimization weight for the optimization algorithm to find the best IIR filter fit in the required frequency region.

When designing a feedback ANC filter, howling is often an issue. Although the Filter PEQ filter is stable with one fit, it may become unstable in other scenarios, such as pressing the ear tip or while inserting the headset into the ear.

Some methods to manage the instability and achieve the best possible ANC performance include:

- Increase the FB filter gain and phase margin.
- Reduce system gain using either of the parameters:
 - Feedback maximum open loop gain.
 - Target gain.
 - Gain offset.
- Howling/instability at high frequencies (above approximately 7 kHz to 8 kHz).
 - Decrease cutoff frequency of LPF to attenuate high frequency amplification.
- Howling/instability at mid frequencies (in approximately 1 kHz to 6 kHz band).
 - Decrease phase lag in filter response. There are two ways to resolve this:
 - Increase LPF and/or smLPF cutoff frequency.
 - Change target filter to have less roll-off at higher frequencies. Even positive sloping magnitude can help in some systems). Less roll-off means less phase lag and more stability. This approach may also require simultaneous reduction in filter gain.
- Identify the howling frequency and add a notch/reduce the gain in that frequency region of the target filter. The notch should not be too aggressive.
- If the howling occurs in the low frequencies, reduce low frequency magnitude response in the target filter or change the **Low Frequency OL unit gain limit** parameter.

Controlling the noise overshoot

Noise *overshoot* (also known as *waterbed* in feedback systems, or *amplification* in feed-forward and hybrid systems) is a side effect of ANC. Noise overshoot is always observed to some degree and depends on the aggressiveness of the ANC performance.

To control the noise overshoot, limit the maximum overshoot to no more than 3 dB, and design an ANC profile that has a small amount of overshoot (3 dB) over a wide bandwidth, rather than a large amount of overshoot (> 5 dB) in a small bandwidth.

Some other methods to mitigate the noise overshoot include:

- Avoid sharp roll-off in the feedback target filter shape. Sharp roll-off produces large phase lag, which decreases the phase margin and generally increases waterbed.
- Reduce the feedback path gain.
- Test the ANC tuning for several different fittings of the device to ensure that the overshoot is not optimized for a single fitting.

Advanced plot options for feedback tuning

For more plot options for feedback tuning, right-click on the amplitude or phase plot of the ANC Filter Designer window. These options include displaying various models loaded and open loop response.

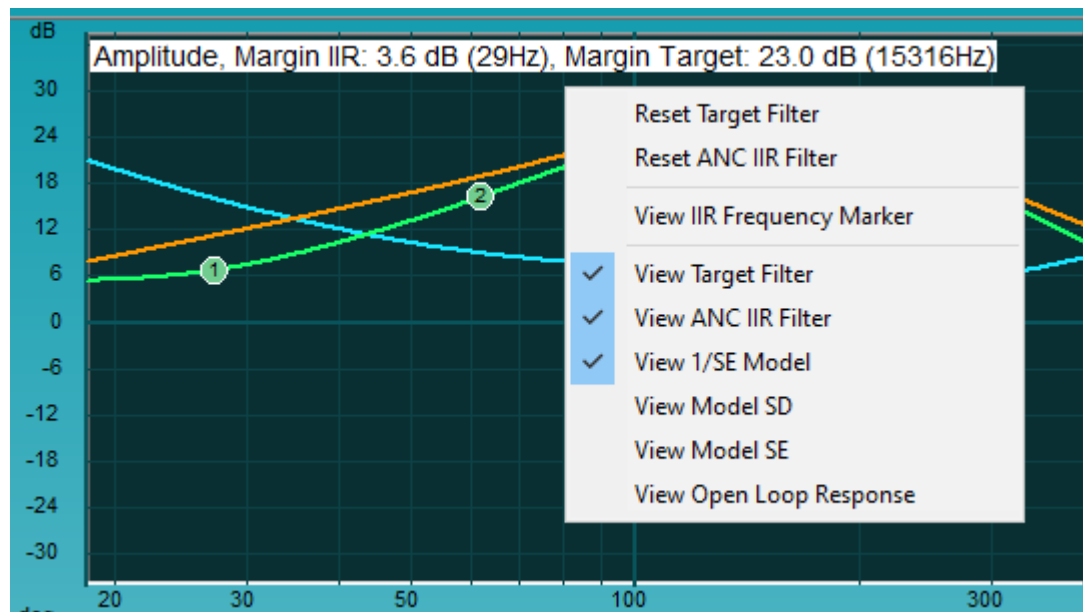


Figure 9-33 Amplitude phase plot of ANC Filter Designer window

9.10 Feed-forward ANC filter design

To design a Feed-forward ANC filter, configure feed-forward ANC, then perform automatic tuning and advanced tuning

The pre-requisites to tuning feed-forward ANC are:

- Configuration settings, see [ANC Filter Designer configuration](#)
- Generate/load reference models (if using Automatic Filter Design mode), see [Generate reference models in Tuning mode](#) or [Generate reference models in Mission mode](#).

9.10.1 Configure feed-forward ANC

To adjust the configuration for the feed-forward path, select the **Feed Forward** tab on the Configuration dialog.

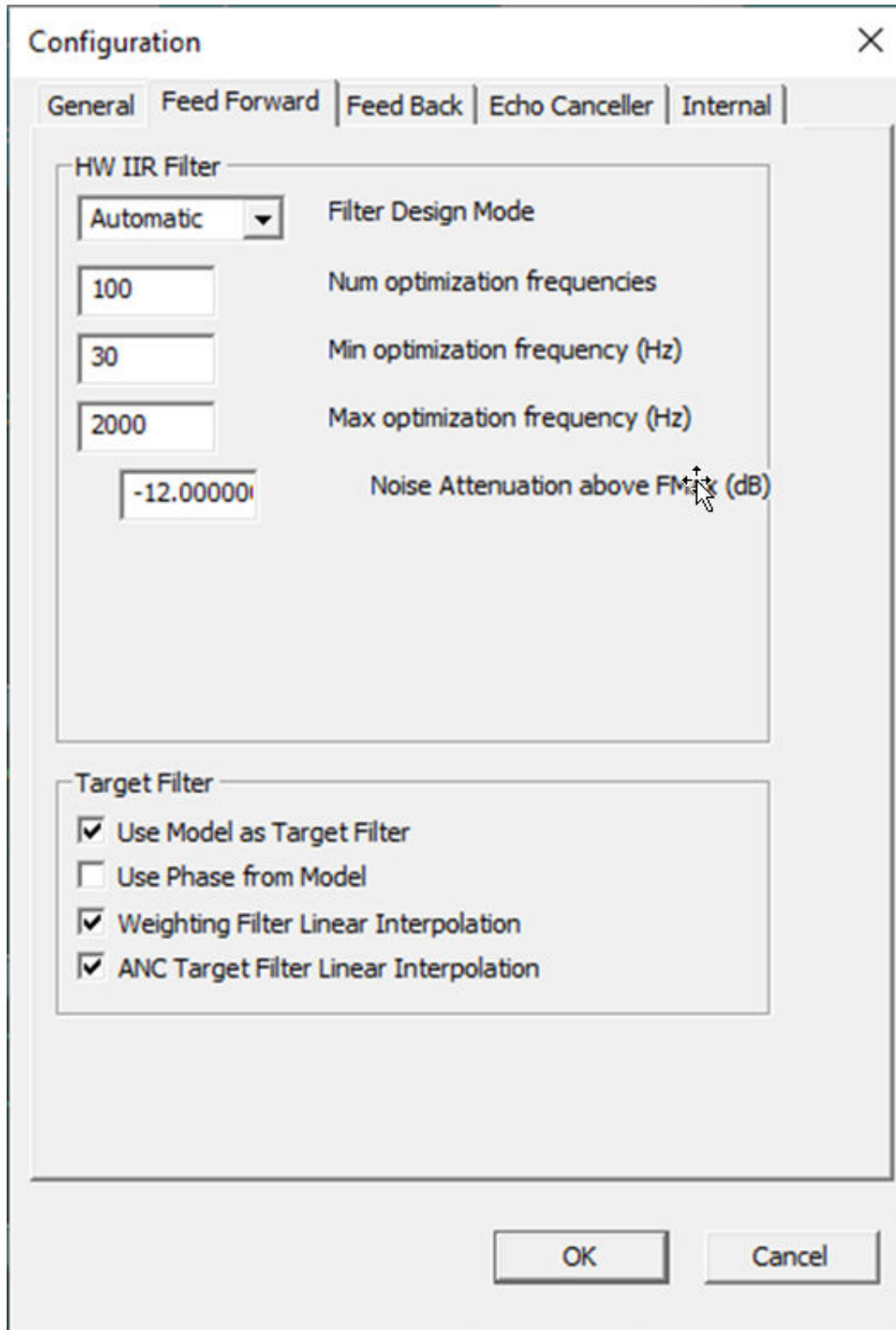


Figure 9-34 Feed-forward path Configuration dialog

9.10.2 Automatic feed-forward tuning

Automatic feed-forward tuning involves generating the feed-forward ANC filter using either the Automatic mode or Automatic Hybrid mode. In the *Automatic mode*, a design feed-forward filter is based on the feed-forward ANC performance defined by the user. In the *Automatic Hybrid mode*, a design feed-forward filter is based on the Hybrid ANC performance defined by the user in the feed-forward design window. A feedback filter must be designed or loaded before using the Automatic Hybrid mode.

To set the Automatic feed-forward tuning mode, see [ANC filter design modes](#).

To generate the feed-forward ANC filter using Automatic mode or Automatic Hybrid mode:

1. Generate and load the PE/PD/SE/SD path models of DUT using the Model menu.
2. Using the PEQ curve in the ANC performance plot, set the target ANC performance.
3. Use the PEQ curve in the optimization weight to emphasize the frequencies for the fitting algorithm to find the IIR filter that closely matches the ANC performance in the frequencies with more weight.
4. Set the Tuning parameters, see [Table 9-6](#).

Table 9-6 Automatic feed-forward tuning parameters

Parameter	Description
Fit HW to target	Start filter optimization algorithm. It finds the IIR filter based on QCC517x ANC hardware that best matches the defined ANC performance (or based on the target amplitude and phase PEQ in Filter PEQ mode) NOTE A button press starts a limited number of optimization cycles. Checking the checkbox next to the button runs the fitting algorithm in a separate thread until the check box is unchecked.
Polarity	If the speaker polarity of DUT is inverted from the reference models that are loaded, select the invert PEQ or invert IIR. The definitions are: ■ Invert PEQ: Apply 180° phase shift to the target PEQ. ■ Invert IIR: Apply 180° phase shift to IIR filter that is written to the device.
Gain	To obtain the best ANC performance, apply the appropriate gain to the IIR filter. If additional gain is needed, apply it through Target gain or Gain offset.
Target gain	Gain of the target PEQ.
Gain offset	Gain offset that is applied to the ANC hardware without modifying the target filter.
LPF	The LPF dropdown can control the noise overshoot, out of band noise aliasing into audio band.

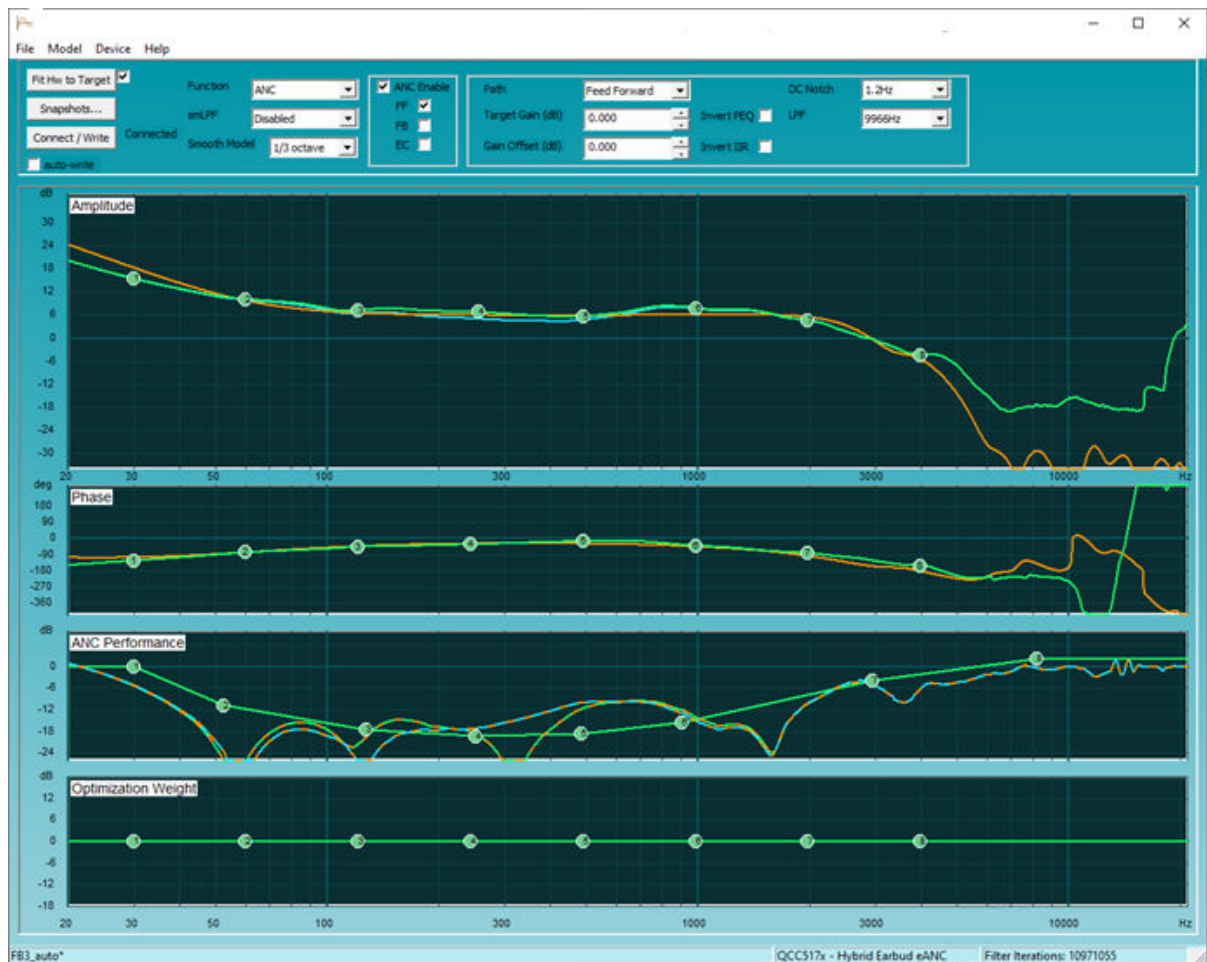


Figure 9-35 Automatic feed-back tuning using Automatic mode or Hybrid mode

NOTE The PEQ curve is activated only if the Automatic mode is selected in feedback ANC configuration settings.

9.10.3 Advanced feed-forward tuning

Advanced feed-forward tuning refers to the fine tuning that is sometimes needed to optimize the ANC performance or to overcome the instability. This includes accounting for fit variation, performance bandwidth, noise overshoot, and adjusting the feed-forward gain.

Fit variation

Feed-forward ANC is more susceptible to fit variations than feedback ANC. Even a slight variation in fit can result in considerable loss of feed-forward ANC performance.

To understand any significant loss in ANC performance, analyze variations in both magnitude and phase response of the target filter (Primary path/Secondary path response) for various headset fittings

Depending on the level of fit variation, testing on various equipment or subjects can also lead to different ANC performance experiences.

ANC performance bandwidth

Feed-forward ANC performance bandwidth depends on the headset. Some headsets have ANC performance bandwidth of more than 2 kHz. In these cases, increase the limits of optimization frequency using the Min and Max optimization frequency options in the Feed Forward Configuration dialog window, see [Configure feed-forward ANC](#).

Noise overshoot

The methods to control noise overshoot are similar to the methods for a feedback filter. For details, see [Advanced feedback tuning](#). For feed-forward filter design, avoid sharp peaks in target filter shape.

Adjusting the feed-forward gain

Adjusting the gain alone of a given feed-forward filter can result in drastic performance variation for depth, bandwidth, and noise overshoot. Ensure that the gains of each headset are fine tuned to achieve the performance achieved with the golden unit.

9.11 Echo canceller filter design

When the feedback ANC filter is enabled, it results in attenuation of low frequencies during music playback. To overcome this, tune the echo canceller.

NOTE This echo canceller IIR filter is in addition to the feed-forward and feedback IIR filters.

The set up echo canceller tuning:

1. Connect a source device to the DUT using A2DP.
2. Play a stimulus, such as white, pink, or sin sweep, through the headset speaker.
3. Select **Echo Canceller** in the ANC Filter Designer from the path.

NOTE Conduct the tuning in a quiet environment, without stimulus from external speakers.

9.11.1 Configure echo canceler ANC

The configuration dialog for the echo canceler path has the same parameters as for the feed-forward path, except that Filter Design mode is fixed to Filter.

Configuration

General | Feed Forward | Feed Back | **Echo Cancellor**

HW IIR Filter

Filter Design Mode: Filter

Num optimization frequencies: 50

Min optimization frequency (Hz): 50

Max optimization frequency (Hz): 2000

Noise Attenuation above FMax (dB): -12.00000

Target Filter

☒ Use Model as Target Filter

☐ Use Phase from Model

☒ Weighting Filter Linear Interpolation

OK Cancel

Figure 9-36 Echo canceler path configuration dialog

9.11.2 Automatic echo canceller tuning

Echo canceller tuning involves minimizing the loss in low frequencies when ANC is turned on vs when ANC is turned off.

To ensure that the music playback spectrum between ANC off vs ANC on is as close as possible when the echo canceller filter is enabled, perform the following tasks:

1. Check the target spectrum:
 - a. Turn ANC off while playing the stimulus signal.
 - b. Get the spectrum of the playback at the reference microphone, see the red curve in [Figure 9-37](#).
2. Enable ANC and obtain the spectrum. This helps determine which frequencies need to be tuned, see the blue curve in [Figure 9-37](#).
3. Adjust the PEQ curve and the optimization weight in the ANC Filter Designer until the spectrum of the ANC on and ANC off are similar, especially in the low frequency region.
4. Toggle between ANC on/off to identify any noticeable differences between ANC on vs ANC off.

NOTE Tuning the echo canceller depends on the SE/SD path, it is independent of feedback ANC filter.

[Figure 9-37](#) shows a spectrum of ANC off, ANC on without echo canceller, and ANC on with an echo canceller filter.

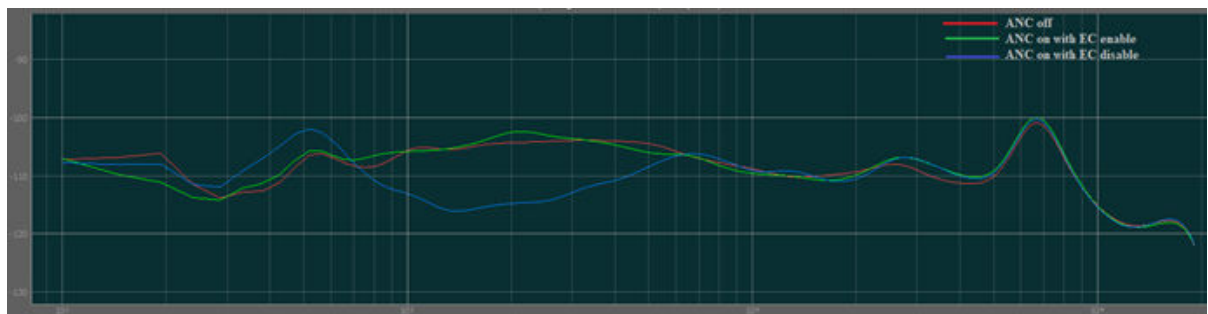


Figure 9-37 Echo canceller filter spectrum

Figure 9-38 shows an example of echo canceller tuning from the ANC Filter Designer.

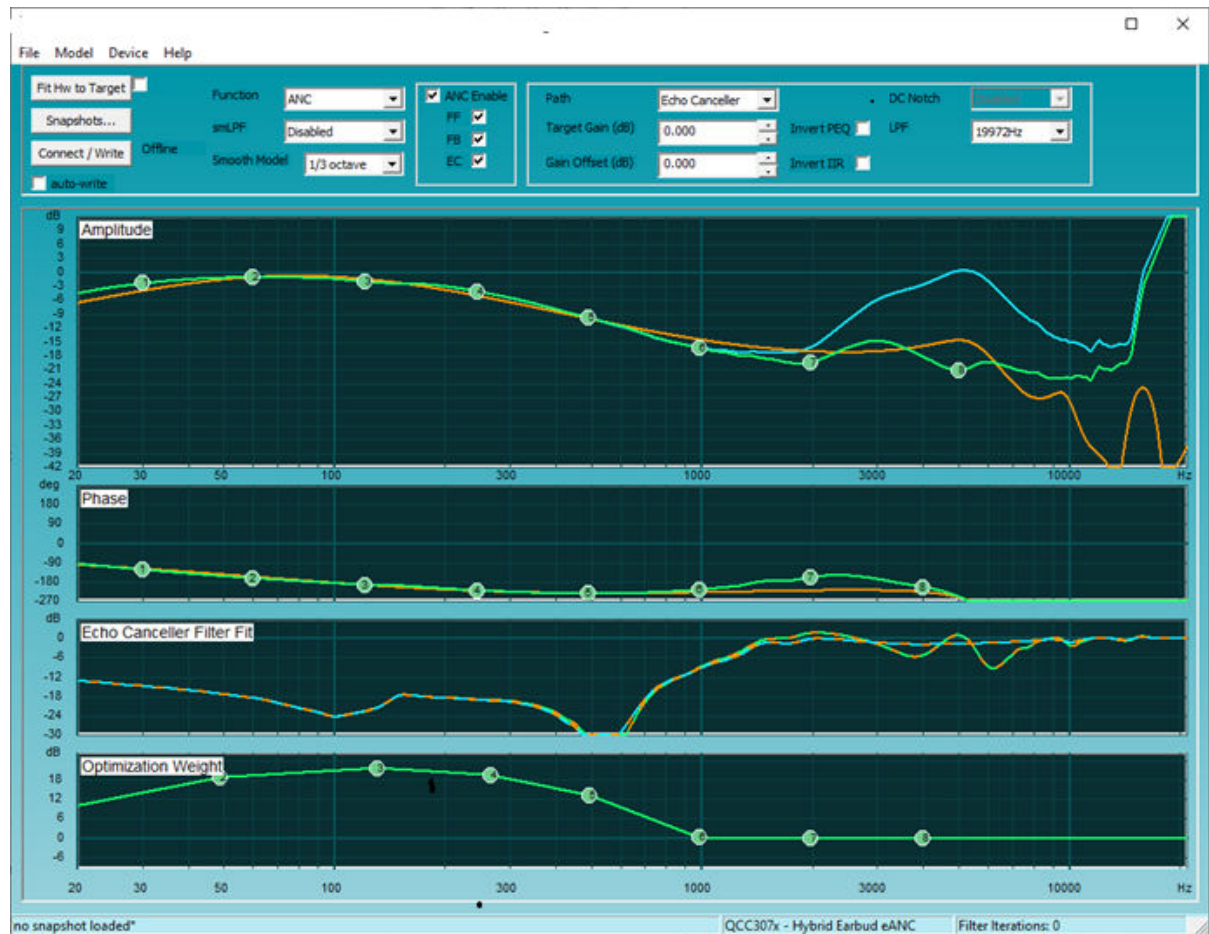


Figure 9-38 Echo canceller tuning example

9.12 Leak-through tuning

In leak-through mode, the feed-forward microphone provides awareness of the tuning environment to the user.

To set up leak-through tuning:

1. From the function drop-down, select **Leak through**.
2. From the path, select **Feed Forward**. Ensure that the polarity (Invert PEQ/IIR) is the opposite of the polarity selected for ANC mode.

RxMix

To mitigate the occlusion effect of the headset, enable the feedback filter in leak-through mode. The feedback filter cancels the user's own voice and also cancels ambient signals.

ANC hardware contains an RxMix path, which keeps the ambient signal unchanged, but cancels the user's self voice. The RxMix path functionality is similar to the echo canceller path when feedback ANC is enabled.

9.12.1 Automatic leak-through tuning

Automatic leak-through tuning involves generating the leak-through ANC filter using either the Automatic mode or Filter PEQ mode.

When performing automatic leak-through tuning, ensure that you mitigate the following:

- Instability
- Noise floor in quiet environments

Instability

Under certain conditions, the feed-forward microphone can pick up signals from the headset speaker, resulting in instability when leak-through mode is enabled. Ensure that the leak-through filter is not so aggressive that it howls while the earbuds are adjusted in the user's ears. This instability can also be controlled using the HCGR capability, which is a part of Adaptive ANC . For more information about the HCGR capability, see [Adaptive Ambient tuning](#).

Noise floor in quiet environments

When leak-through mode is enabled in a quiet environment, the noise floor may be amplified and become audible, depending on the aggressiveness of the filter. To mitigate this noise floor effect, use one of the following methods:

- Reduce the filter gain, which also reduces the leak-through mode performance.
- Use the Quiet Mode functionality of Adaptive ANC, which reduces the feed-forward path gain to a predetermined value when Quiet mode is detected.

9.13 ANC Filter Designer plot view

Plot windows provide fast, interactive graphics for displaying data. The ANC Filter Designer has options to manipulate the plot legend and view of the plot.

Plot legend

Table 9-7 ANC Filter Designer plot legend

Plot	Description
Amplitude	▪ Blue: Model ▪ Green: Target PEQ ▪ Orange: Hardware IIR, including all hardware filters
Phase	▪ Blue: Model ▪ Green: Target PEQ ▪ Orange: Hardware IIR, including all hardware filters
ANC performance	▪ Orange-Green: Simulated ANC performance Hardware IIR vs. Target PEQ ▪ Green: ANC target
Weight	Green: Weighting curve for IIR optimization

Y-Zoom and Y-Scroll

To enable a Y-Zoom, move the mouse over the y-axis labels and use the mouse wheel to zoom in.

To enable a Y-Scroll, press the left mouse button in the plot window, and move up and down.

Context menu

To select the context menu, press the right mouse button on either of the plot windows.

9.14 Save the tuning and workspace

Tuned filters can be saved for later use, or to capture the current state of the GUI.

To save a filter after it has been designed, use the following options from file menu:

- [Snapshots](#)
- [Load/save HTF file](#)
- [Export the IIR filter](#)

9.14.1 Snapshots

Snapshots capture the ANC Filter Design state while working on it, or to return to a previous ANC Filter Design state. Snapshots also help to simplify switching between different filters, for comparison.

Figure 9-39 shows the snapshot dialog in the ANC Filter Designer.

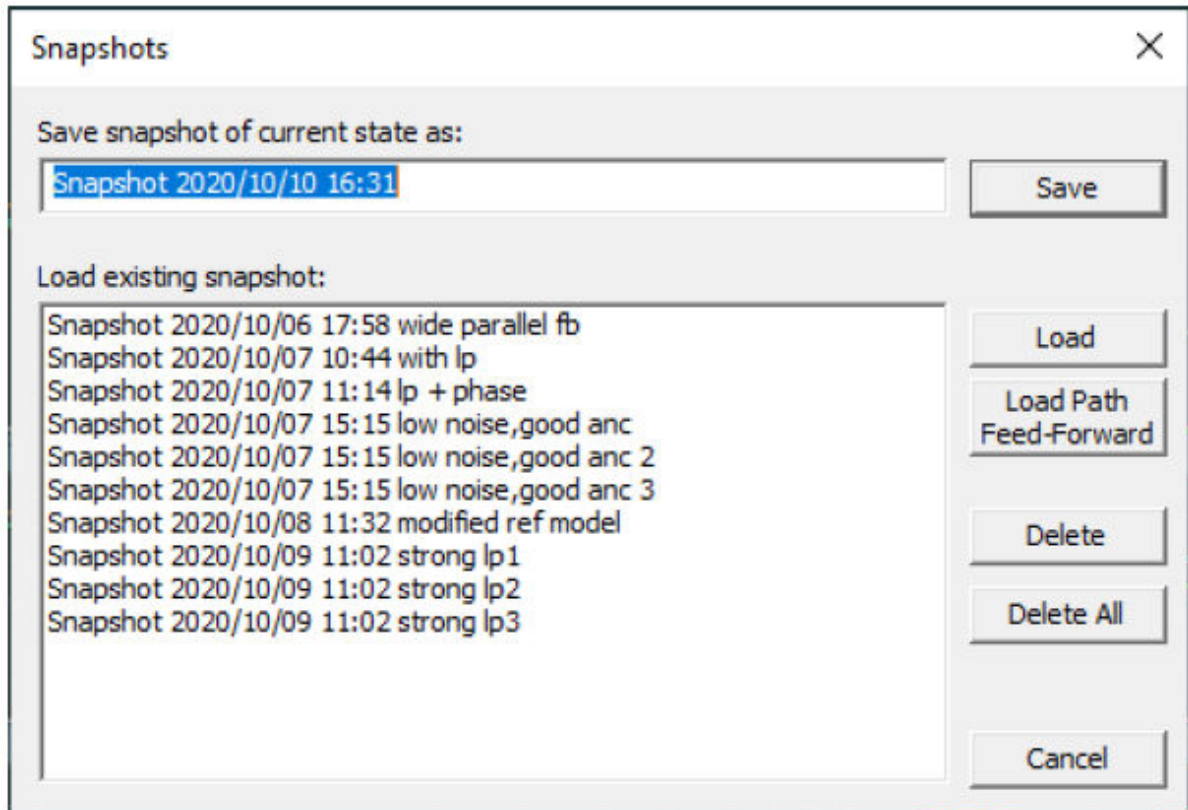


Figure 9-39 Saving a ANC Filter Designer Snapshot

Create snapshots

The upper area of the dialog allows to create new snapshots containing the current state. The name of the snapshot is pre-filled with the current date and time, but it can be modified or extended to add noteworthy characteristics. By selecting an existing snapshot in the list in the lower part, that name is copied to the save name. It could be left as-is to overwrite the existing snapshot or amended with further information.

To create the snapshot, click **Save**. A confirmation dialog displays when a snapshot with the required name already exists.

Load snapshots

The lower part of the dialog displays the list of captured snapshots. After a snapshot is selected in the list, click **Load** to load that exact state to further work with it.

To only load the part of the snapshot that is related to the path that is currently displayed on the main screen, use a second button. Other paths of the current filter design are left untouched. The button

indicates which path would be replaced (**Feed-Forward** > **Feed-Back** > **Echo- Cancel**) when the button is clicked.

Delete snapshots

To remove a snapshot, select it from the list in the Snapshot dialog and click **Delete**.

NOTE Use the **Delete All** button with caution, because it removes all snapshots from the workspace.

The status bar displays information about a Snapshot.

Figure 9-40 shows an example of a snapshot status bar, displaying the name of the snapshot on the left, device information in the middle, and the number of filter optimization iterations on the right.

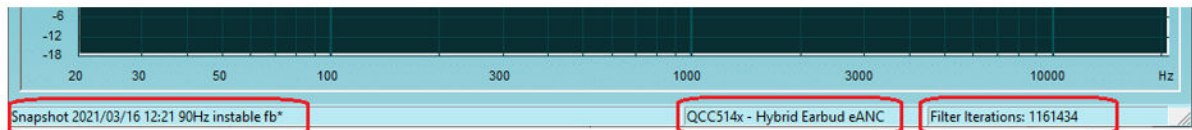


Figure 9-40 Snapshot status bar

9.14.2 Load/save HTF file

ANC tuning parameters can be loaded from and stored to an HTF file.

Figure 9-41 shows the dialog to load or save an HTF file.

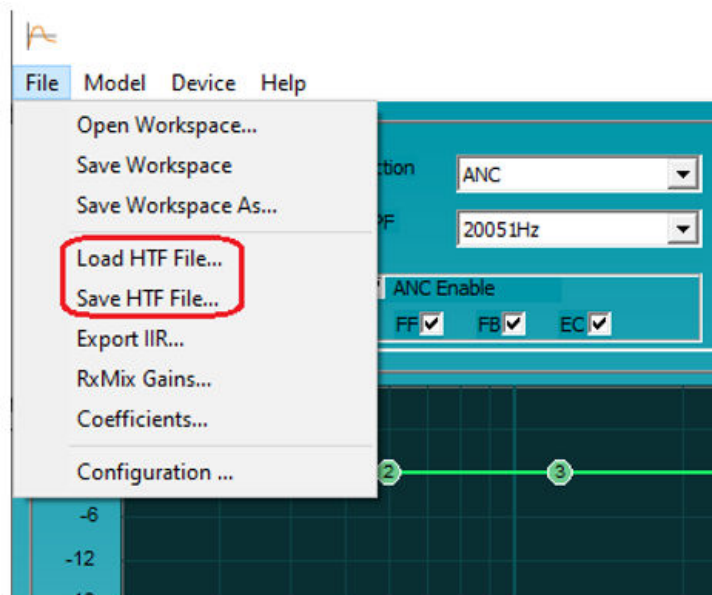


Figure 9-41 Load/save ANC tuning parameters to an HTF file

NOTE When a HTF file is loaded, the hardware filter settings are updated in the ANC Filter Designer. The reference and PEQ design curves are not affected.

If an HTF file contains parameters for several ANC modes (UCIDs), select the intended mode to read from the dialog that displays a dialog displays.

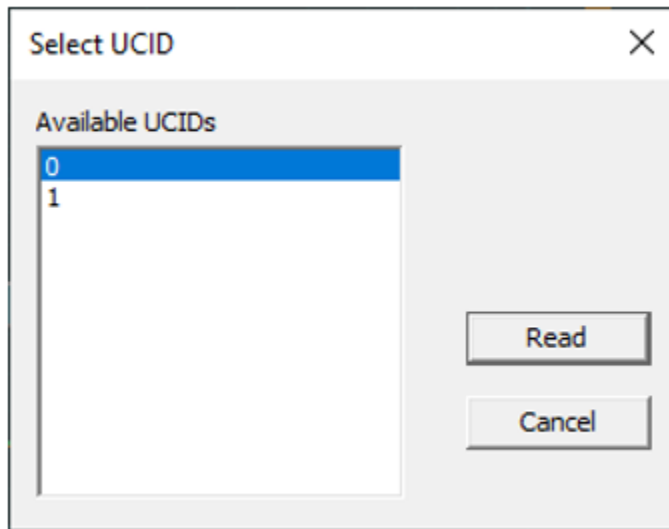


Figure 9-42 Select UCID to read

When writing the current ANC tuning to an HTF file, set the filename and UCID in the Save dialog.

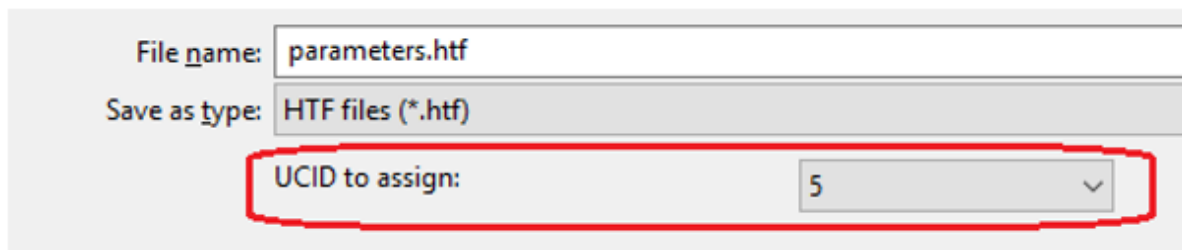


Figure 9-43 Set UCID in HTF File Save Dialog

9.14.3 Save workspace

The workspace file (*.pws) contains all configurations, filter parameter, snapshots, and paths to reference files. To restore a tuning session, save and load a workspace file.

Figure 9-44 shows the file menu to open and save Workspaces.

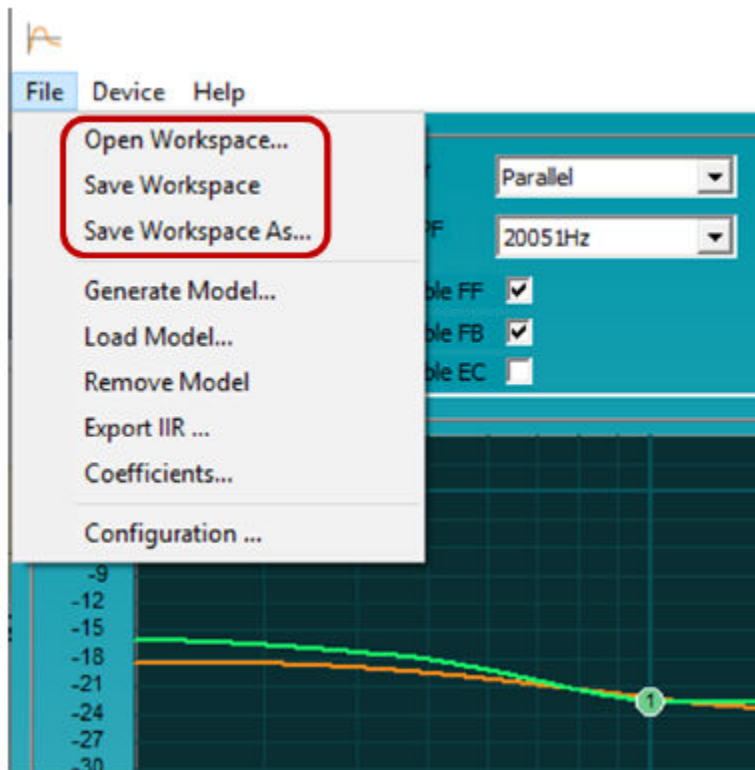


Figure 9-44 Open and Save ANC Workspaces dialog

9.14.4 Read and write from persistence

The ANC Filter Designer can read and display configurations that are available in the persistent device storage. The ANC Filter Designer can also be written to persistent storage.

Read persistence

Reading only updates the IIR filter and displays its response. The reference and PEQ design curves are not affected.

An HTF file with the configuration is also written to file `anc_modeX.htf`.

Write persistence

Writing the ANC Filter Designer to persistence can be helpful to run tests without the ANC Filter Designer being connected, and when using the application to switch between different ANC modes.

Figure 9-45 shows the dialog that selects the ANC mode to be written.

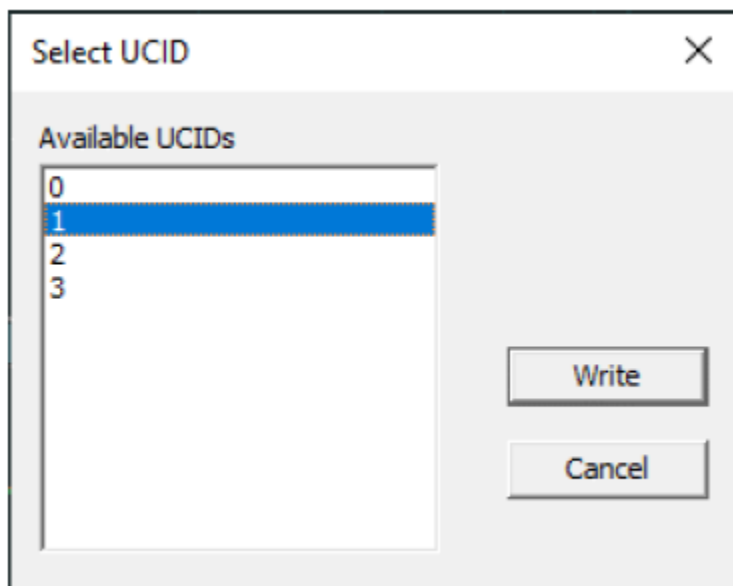


Figure 9-45 Selecting an ANC mode to be written

The number of selectable modes (called UCIDs) varies, depending on the number of ANC modes that are configured in the application software.

An HTF file with the configuration is also written to file `anc_modeX.htf`.

9.14.5 Export the IIR filter

The frequency response of a designed IIR filter can be exported to file.

Figure 9-46 shows the dialog to export the IIR filter.

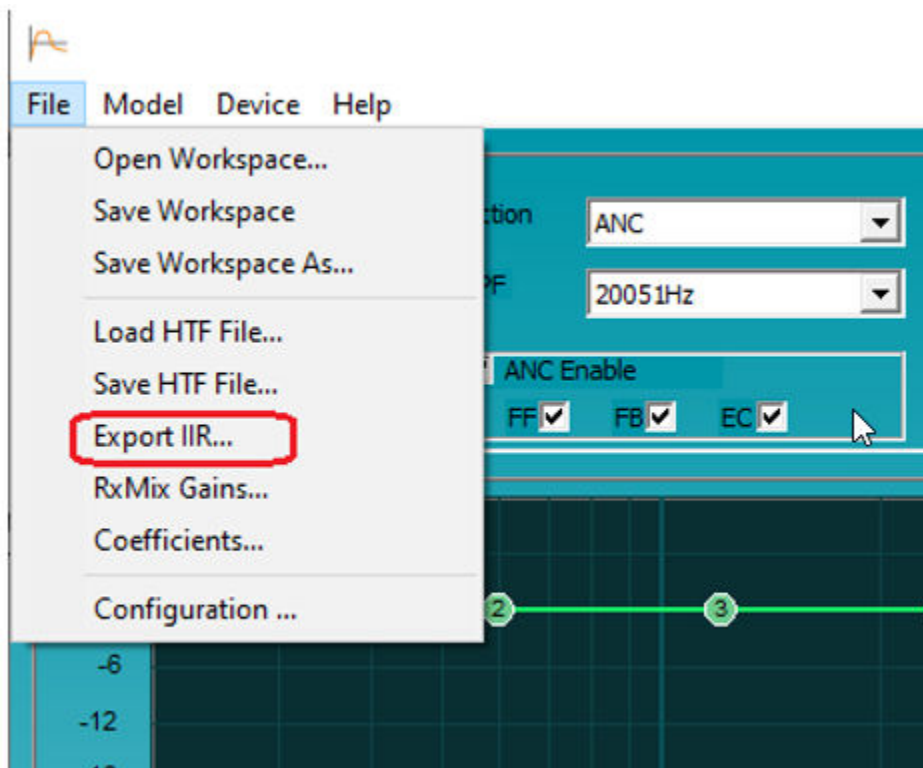


Figure 9-46 Export IIR

9.15 Configure frontend gains

The Frontend Gains dialog displays the microphone and speaker gain settings as part of the overall ANC settings on a device.

Figure 9-47 shows the Frontend Gains dialog.

Label	Hex Value	dB Value	Unit	Checkbox
FFa Mic Gain L	0x0000	0.00	dB	☒ Dig mic clock rate FFa L
FFb Mic Gain L	0x0384	15.00	dB	☒ Dig mic clock rate FFb L
Speaker Gain L	0x0000	0.00	dB	
FFa Mic Gain R	0x0000	0.00	dB	☒ Dig mic clock rate FFa R
FFb Mic Gain R	0x0384	15.00	dB	☒ Dig mic clock rate FFb R
Speaker Gain R	0x0000	0.00	dB	

Notes:
 Current gains are read from device during connection.
 Live change is possible in Tuning Mode, but not when using TestEngine.
 Writing to PSkey has no immediate effect. Values might be used later for initialization during startup.
 Gains must also be consistent with other uses like cVc.

Ok Cancel

Figure 9-47 Frontend Gains dialog

In general, the frontend gains should be configured consistently in the image build. When they are only changed relative to ANC, issues might arise from other, concurrent uses.

However, the frontend gains can be changed and written to persistent device storage as part of the ANC configuration. This enables them to be used for initialization during start up.

In Tuning mode, the effect of changed microphone gains can be observed live on the device after writing the ANC configuration to the device.

10 Production-line calibration

Factory or production-line calibration involves tuning QCC517x and QCC307x chipset ANC path gains on the production-line to account for sensitivity variation in the microphones and speakers.

Factory calibration must be performed on fully assembled headsets. Although factory calibration typically involves tuning only gain blocks, the same functionality is provided for tuning all other ANC block parameters.

Qualcomm recommends adjusting the gains for each ANC path (feed-forward, feedback, and echo-canceller) due to the variance in speaker response and microphone sensitivity, and account for manufacturing tolerances. The calibration process involves tuning gain parameters in real-time on an acoustic test fixture and saving the final calibration parameters to flash.

Factory calibration mainly involves changing gain parameters to meet a predefined performance target identified during the development process. In contrast, the development process involves iteratively designing filters and measuring the performance of various ANC filters and other ANC block parameters.

10.1 Tuning setup and process

Figure 10-1 shows the high-level process for deploying ANC to production.

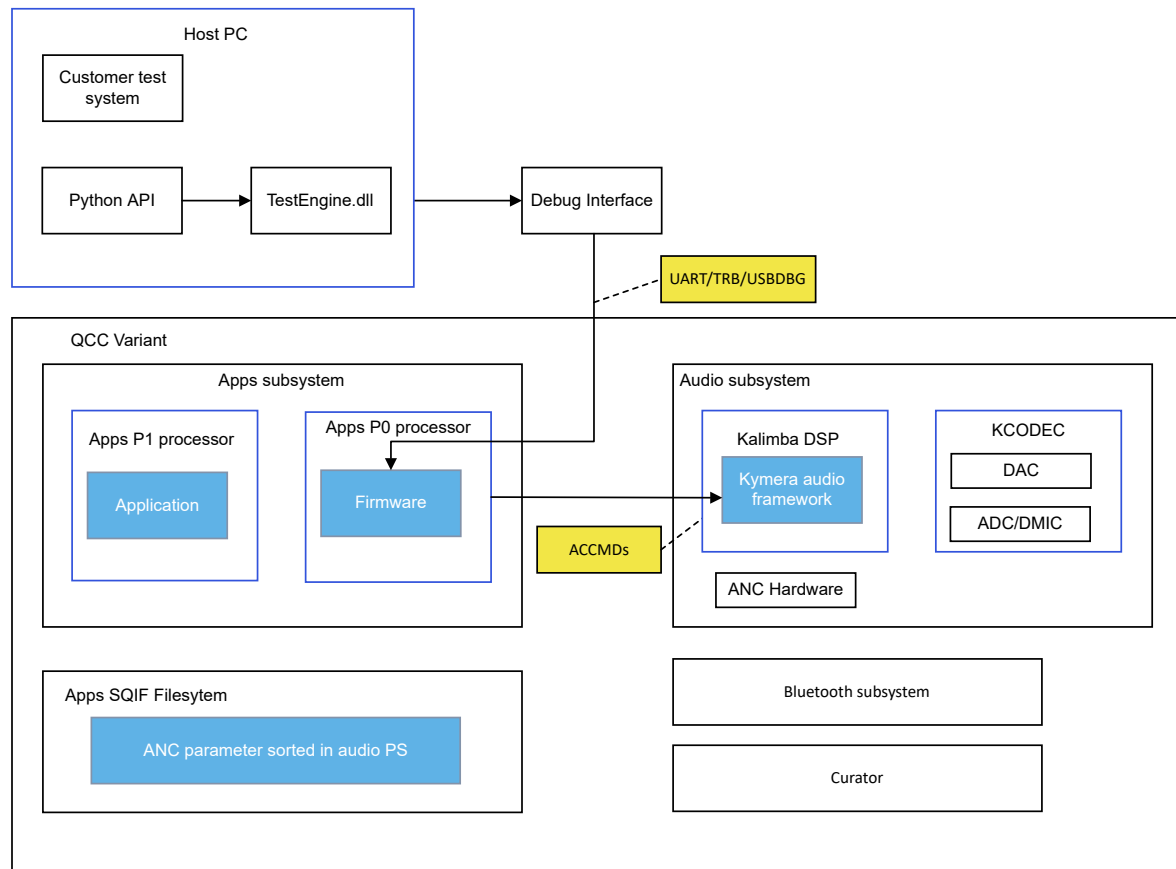


Figure 10-1 Setup: deploying ANC to production

The high-level process for deploying ANC to production involves:

1. Generating tuning parameters using the ANC filter designer tool. The ANC filter designer tool provides the HTF file as an output, which is added to the final product image.
2. Developing production-line test applications using the TestEngine API and either the USB or TRB interface.
3. Assembling each unit and testing ANC on a special test setup.

4. Determining calibrated gain parameters through an iterative process of measurement and final gain adjustments.
5. Writing final calibrated parameters to device persistence. While any parameter can be adjusted on the production-line, the fine gain for each path is typically the only parameter that needs adjustment.
 - Feed-forward gain left
 - Feed-forward gain right
 - Feedback gain left
 - Feedback gain right
 - EC gain left
 - EC gain right

10.2 Hardware and software requirements for deploying ANC

The hardware and software requirements for deploying ANC to production are listed in [Table 10-1](#).

Table 10-1 Hardware and software requirements

Hardware	Software
Test fixture with internal microphones	TestEngine ANC API to communicate with device
Speaker to generate noise	The audio measurement system, for example, Sound Check, CRY, or customer-developed system
Enclosed or semi-enclosed acoustic chamber	A top-level application, which includes testing logic, GUI, measurement calculation, pass or fail limit checking mask
Host PC	—
USB sound card	—
USBDBG or TRB or Bluetooth connection to DUT	—

10.3 Wireless calibration

Wireless calibration enables ANC fine gain production-line calibration on a sealed earbud over UART transport.

The Wireless calibration provides a reference code for ANC gain parameters after plastic calibration. This value must be used in sync with customer-defined audio measurement setup and tools.

FOR IMPORTANT ADVISORY NOTE ON MANDATORY ACTION, SEE [Security advisory](#).

10.3.1 Security advisory

A new security PS key is added to limit access and lock calibration on the production-line after the required ANC gain calibration is achieved.

NOTE It is mandatory to disable UART access post calibration using AT +ANCRESETCALMODE on the production-line, which resets the security PS key,

thereby locking calibration permanently. Failing to do this could expose the product to the risk of de-tuning set (ANC gain) parameters.

10.3.2 ANC test setup for fine tuning

To set up ANC for fine tuning, you require a QCC512x evaluation board (CG793) or newer with a Communicator board and a USB serial adapter between the host and device.

Device setup

To set up the ANC device:

1. Open the earbud workspace from the following path (shown as example):

```
<ADK path>\apps\applications\earbud\ qcc512x_rom_v21\CG793\earbud.x2w
```

If support needs to be added for other platforms, add the following files to the earbud.x2p under the selected platform:

- av_headset_anc_prod_test.c/h
- av_headset_uart.c/h
- av_headset_at.c/h
- av_headset_at_anc.c/h

2. Enable the following defined in the earbud project properties under DEFS:

- ENABLE_ANC_PRODUCTION_TEST
- ENABLE_UART
- ENABLE_PRODUCTION_AT_COMMANDS

3. Ensure that subsys7_psflash.htf sets USR0 PS Key to TRUE (USR0 = [01 00]). This key is used as a Security PS key for production tests to enable and lock calibration after done.
4. Set the ANC microphone configuration within the <ADK path>\apps\applications\earbud \ av_headset_config.h file. Depending on the type of board that is used in the setup, set the parameters in the file accordingly. Make sure the anc_tuning_config.htf file is in sync with the mode settings in av_headset_config.h.

Feed-forward ANC example:

```
/*!@{ @name ANC configuration for feed-forward mode */
#define appConfigAncPathEnable()          (feed_forward_mode_left_only)
/* Disable ANC tuning functionality */
#define appConfigAncTuningEnabled() ( FALSE)
/* Use microphone 0 for ANC feed-forward */
#define appConfigAncFeedForwardMic()      (0)
/* Don't use microphone for ANC feed-back */
#define appConfigAncFeedBackMic()         (NO_MIC)
```

Feedback ANC example:

```
/*!@{ @name ANC configuration for feed-back mode */
#define appConfigAncPathEnable()          (feed_back_mode_left_only)
/* Disable ANC tuning functionality */
#define appConfigAncTuningEnabled()      (FALSE)
```



```

/* Don't use microphone for ANC feed-forward */
#define appConfigAncFeedForwardMic()      (NO_MIC)
/* Use microphone 1 for ANC feed-back */
#define appConfigAncFeedBackMic()         (1)

Hybrid ANC example:

/*!@{ @name ANC configuration for hybrid mode */
#define appConfigAncPathEnable()           (hybrid_mode_left_only)
/*! Disable ANC tuning functionality */
#define appConfigAncTuningEnabled()        (FALSE)
/* Use microphone 0 for ANC feed-forward */
#define appConfigAncFeedForwardMic()      (0)
/* Use microphone 1 for ANC feed-back */
#define appConfigAncFeedBackMic()         (1)

```

5. Build and deploy on the hardware.

Host scripts

Create the required host stubs to send the AT commands and receive the responses.

10.3.3 ANC production calibration configuration

For production-line calibration with a sealed headset, ANC gains are fine-tuned based on variations in plastics and transducer tolerances of the microphone and speaker. As a prerequisite, ensure that the ANC initial filter design is done in the lab to set the filter coefficients and identify the optimum configuration. To design ANC filters, use the QACT tool over USB or TRB terminal.

During calibration, the sealed headset communicates over UART with a host or charger case depending on the configuration. This is a standard UART over separate Tx and Rx lines. The communication interface for fine-tuning uses the AT-command protocol. For more information, see the *Device Test Service User Guide* document.

ANC production calibration set up

The ANC production calibration setup uses the additional pin (UART) on the charger case which communicates with the earbuds and a host PC through a low-cost microprocessor that performs the UART/USB conversion, as shown in [Figure 10-2](#).

The AT commands mapping can happen either in the host or in the low-cost MP.

The following combinations are possible:

- AT commands mapping in host and transport conversion in low cost MP.
- Both AT commands mapping and transport conversion in low cost MP.
- Both AT commands mapping and transport conversion in host and bypassing the Charger case.

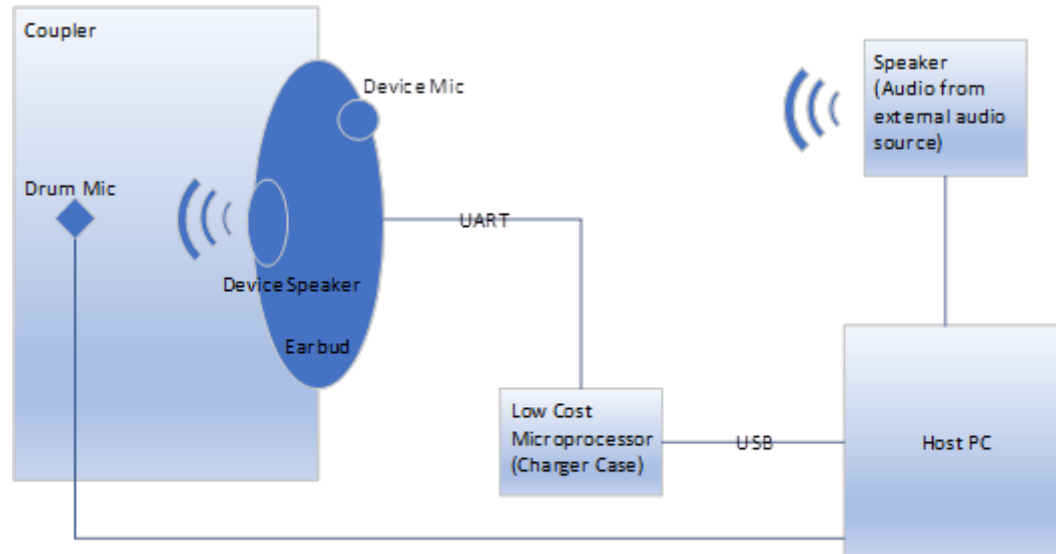


Figure 10-2 ANC production calibration setup

Configuration

The ANC production-line calibration involves calibrating individual earbuds. During calibration, the audio is played from an external noise source triggered by the host PC.

The ANC active attenuation is measured at a microphone (drum mic) inside the coupler (ear simulator). The ANC fine gain can be increased or decreased to match the target attenuation curve specified by the golden tuning.

10.3.3.1 Storing device-specific ANC fine gain

The device-specific ANC Fine Gain is calibrated in the production line calibration for each instance and path.

An interface is available to overwrite the “Golden” ANC Fine Gain (stored in a PSkey) through the DTS module. Each ANC mode and each path has a independent ANC fine gain value. However, when performing a PS-config update (through OTA) new values might overwrite the device-specific ANC fine gain values that were calibrated in the production line calibration.

This feature preserves the device-specific calibrated values to be used after PS config updates.

When storing device-specific ANC fine gain:

- Clear USR13 before any factory calibration action.
- In AHM, select the **Disable FF gain redistribution** option for the calibration mode.
- Create wrappers to read ANC gains based off configuration. If AANC is included in the build config, gains are read from the AANC capability, else the gains are read through the ANC library. These wrappers can be called in the DTS module.

Limitations

It is not possible to store gain parameters in HTF or human-readable format.

Assumptions

For each ANC mode, three fine gain values, *FFA*, *FFB*, and *FB* are calibrated.

However, the delta between the calibrated fine gain value and the golden fine value must be the same across all modes, and therefore, it is necessary to store just three fine gain values, one for each path.

Storing calibrated Delta of Fine Gain into device storage

During storage, the corresponding Fine Gain value from the PSkey (specific for each ANC mode and path) is read. Then the two Fine Gain values (golden gain and write Fine Gain from `ancWriteFineGain()`) are converted into the fixed-point format and the delta between the calibrated fine value and the golden value is stored in a new user Pskey.

Figure 10-3 shows how the calibrated delta of Fine Gain is stored in the device storage.

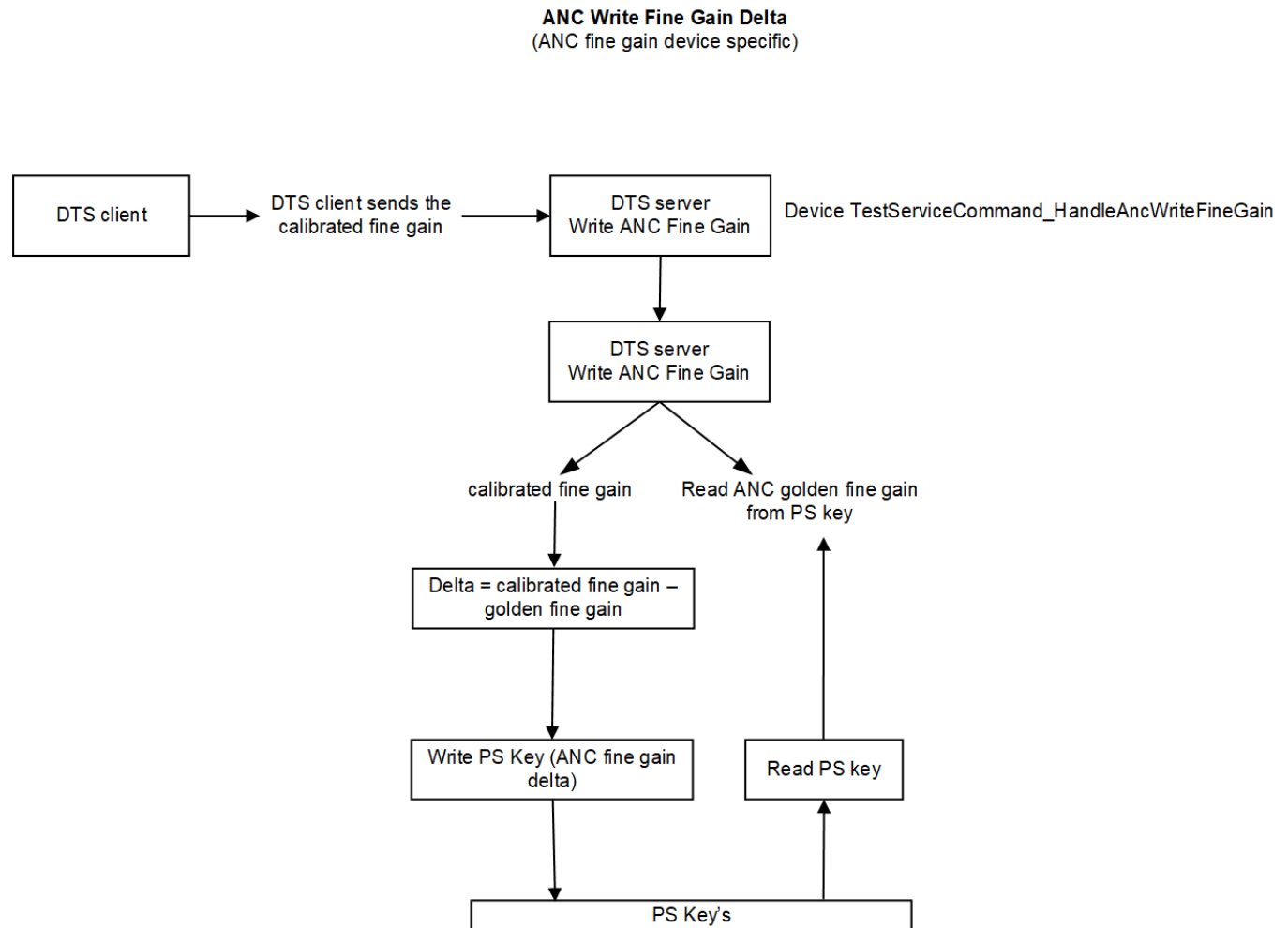


Figure 10-3 ANC Write Fine Gain Delta

Populating device specific ANC Fine Gain values

During boot time or when changing the ANC mode:

1. The ANC configuration is populated into the ANC hardware and software.
2. The golden ANC configuration from the PSkey into an ANC configuration data structure is read and then device-specific data is populated.

The delta of the fine-tuning calculated in the production-line is added to the golden value read from the PSKey. The coarse value must be adjusted if the new Fine Gain value is greater than 6.0 dB or smaller than -6.0 dB.

Gains mapping

Table 10-2 lists the mapping of coarse value to dB and Q6.9 format.

Table 10-2 Coarse value to dB and Q6.9 format mapping

Gain Shift unsigned int	Gain Shift signed int	DB scale	16-bit Q6.9 format
0	0	0	0
1	1	6	3065
2	2	12	6130
3	3	18	9195
4	4	24	12260
5	5	30	15325
6	6	36	18390
12	-4	-24	-12328
13	-3	-18	-9246
14	-2	-12	-6164
15	-1	-6	-3082

10.3.3.2 ANC AT commands for Fine Gain calibration

The ANC AT commands for Fine Gain calibration exposes APIs in the form of custom AT command to allow the host application to calibrate ANC gain values on the device to adjust ANC performance.

The device expects the AT commands suffixed with a <CR><LF> (Carriage Return Line Feed).

Table 10-4 and Table 10-3 list the AT commands used between the device and the host.

Table 10-3 AT commands to the device

Name	Command	Notes
ANC Enable	AT+ANCENABLE=<mode>	Enables ANC on the device for the request mode. <mode>ANC mode from 1 to 10. Response: OK or ERROR if not authorized.
ANC Disable	AT+ANCDISABLE	Disables the ANC mode on the device. Response: OK or ERROR if not authorized.
ANC Read Fine Gain	AT+ANCREADFINEGAIN=<mode>, <gain_path>	Requests to read the Fine Gain setting from the device for a specific mode and gain path. This is the golden config gain (read from device persistent store) plus the delta gain, if configured Only instance 1 gain is returned in response. The same value is configured for both instances during tuning <gain_path> 1 FFA PATH

Table 10-3 AT commands to the device (cont.)

Name	Command	Notes
		2 FFB PATH 3 FB PATH Response: +ANCREADFINEGAIN:<gain_value> OK or ERROR if not authorized or unsupported <mode> or <gain_path>
ANC Set Fine Gain	AT+ANCSETFINEGAIN=<mode>,<gain_path>, <gain_value>,<gain_value>	Sets the Fine Gain value on the device for a specific mode and gain path in the Audio subsystem. This is a temporary value to check the frequency response against the gain settings. By default, ANC instance 0 gain is set. For Enhanced ANC, instance 1 is also set with the same value. The mode is the same as set during ANC enable. <gain_path> 1 FFA PATH 2 FFB PATH 3 FB PATH <gain_value> 0-255 Response: OK or ERROR if not authorized or unsupported <mode> or <gain_path>
ANC Write Fine Gain	AT+ANCWRITEFINEGAIN=<gain_path>,<gain_value>	Writes the Delta gain (the difference between the golden config gain and the configured gain_value in ANCWRITEFINEGAIN) into the Device Delta PS key, for a specific mode and gain path. This is a persistent value set in the device once the desired gain setting is achieved. ANC on Enable or Mode change picks up the configured golden gain along with the delta gain to configure the hardware. The Delta gain is common for both instances for specified paths and also for different modes of ANC. <gain_path> 1 FFA PATH 2 FFB PATH 3 FB PATH <gain_value> 0-255 Response: OK or ERROR if not authorized or unsupported <mode> or <gain_path>
Reset Calibration	AT+ANCRESETCALMODE	Resets the security PS key in the device and permanently disables the ANC calibration mode. This is a mandatory step after calibration.

Table 10-3 AT commands to the device (cont.)

Name	Command	Notes
ANC Read Coarse Gain	AT+ANCREADCOARSEGAIN=<mode>,<gain_path>	Requests to read the coarse gain setting from the device for a specific mode and path. Only instance 1 coarse gain is returned in response. The same value is configured for both instances. +ANCREADCOARSEGAIN:<gain_value>OK or ERROR if not authorized or unsupported <mode> or <gain_path>
ANC Read Audio PS key for an instance	AT+ANCGETPSKEY = <ANC Audio PS key ID>, <ANC Instance>	Requests to read the entire contents of an ANC instance from the ANC audio PS key. The length of data read from the PS key is dependent on the platform. <ANC instance> 0 instance_0 1 instance_1
ANC Write Audio PS key for an instance	AT+ANCSETPSKEY= <ANC Audio PS key ID>, <ANC Instance>,<Value>	Requests to write the entire contents of an ANC instance to ANC audio PS key (persistent). The length of data read from the PS key is dependent on the platform. <ANC instance> 0 instance_0 1 instance_1
ANC Read Fine Gain for both instances	AT+ANCREADFINEGAINDUAL = %d:mode , %d:gainpath Deprecated post ADK21.1	Reads the Fine Gain for both ANC instances from ANC PS key. This does not consider the delta gain. This command is deprecated from ADK21.1 onwards. Qualcomm recommends using ANCREADFINEGAIN command instead.
ANC Set Fine Gain for both instances	AT+ANCSETFINEGAINDUAL = %d:mode , %d:gainpath , %d:instance0gain , %d:instance1gain Deprecated post ADK21.1	Sets the Fine Gain for both ANC instances dynamically for both instances. This command is deprecated from ADK21.1 onwards. Qualcomm recommends using ANCSETFINEGAIN command instead.
ANC Write Fine Gain for both instances	AT+ANCWRITEFINEGAINDUAL = %d:mode , %d:gainpath , %d:instance0gain , %d:instance1gain Deprecated post ADK21.1	Writes the fine gain for both ANC instances to the ANC PS key. This does not consider the delta gain. This command is deprecated from ADK21.1 onwards. Qualcomm recommends using ANCWRTFINEGAIN command instead.

Table 10-4 AT commands from the device

Name	Response	Notes
Ok	<CR><LF>OK<CR><LF>	Sent to indicate the successful receipt of a supported command and correct associated parameters.

Table 10-4 AT commands from the device (cont.)

Error	<CR><LF>ERROR<CR><LF>	Sent to indicate an error in response to the receipt of a command due to unsupported, incorrect parameters, or not in the right mode.
ANC Read Gain Response	<CR><LF>+ANCREADFINEGAIN: <gain_value><CR><LF>	Response to the ANC Read Fine Gain request containing the Fine Gain. <gain_value> 0-255.
ANC Read Coarse Gain Response	<CR><LF> +ANCREADCOARSEGAIN: <gain_value><CR><LF>	Response to the ANC Read Coarse Gain request containing the coarse gain <gain_value> uint16 value.
ANC Read PS key for instance data	<CR><LF>+ ANCGETPSKEY: <ps_key_value><CR><LF>	Response to the ANC Read PS key for instance data. <instance_data> Length of the instance data is dependent on the platform used.

ANC AT command limitations for fine gain calibration

The AT command parser supports only the commands defined in [Limitations](#). It is expected to wait for the given AT command response before submitting the next command. The commands cannot be chained. The Test command (=?) and Read command (?) are not supported.

Sequence of AT events

Figure 10-4 shows a typical sequence of AT events.

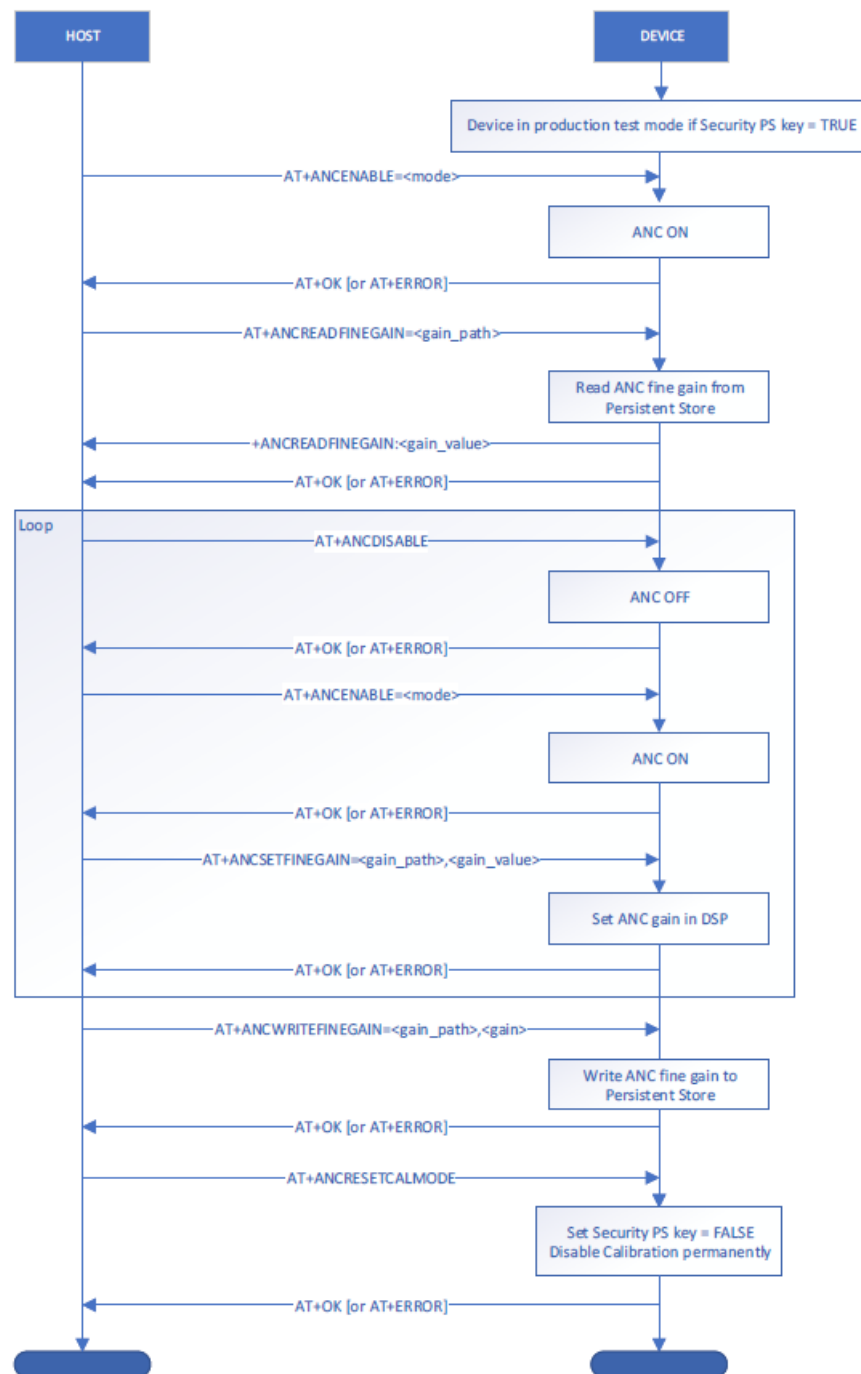


Figure 10-4 AT events sequence

10.3.3.3 Device state

The ANC device state indicates whether the device is in test mode or mission mode.

The Earbud Application goes into production test mode by default if the Security PS key is enabled in the device. This is done through `subsys7_psflash.htf` where USR0 PS Key is required to be set to TRUE, for example, USR0 = [01 00] and deployed in the image.

The test mode configures the UART transport to receive the AT commands from the host. ANC calibration using AT commands can be performed in this mode.

After determining the desired ANC gain calibration, disable this mode. It is mandatory to reset the security PS key in the device and permanently disable ANC calibration mode once calibrated.

Figure 10-5 describes the state of the device.

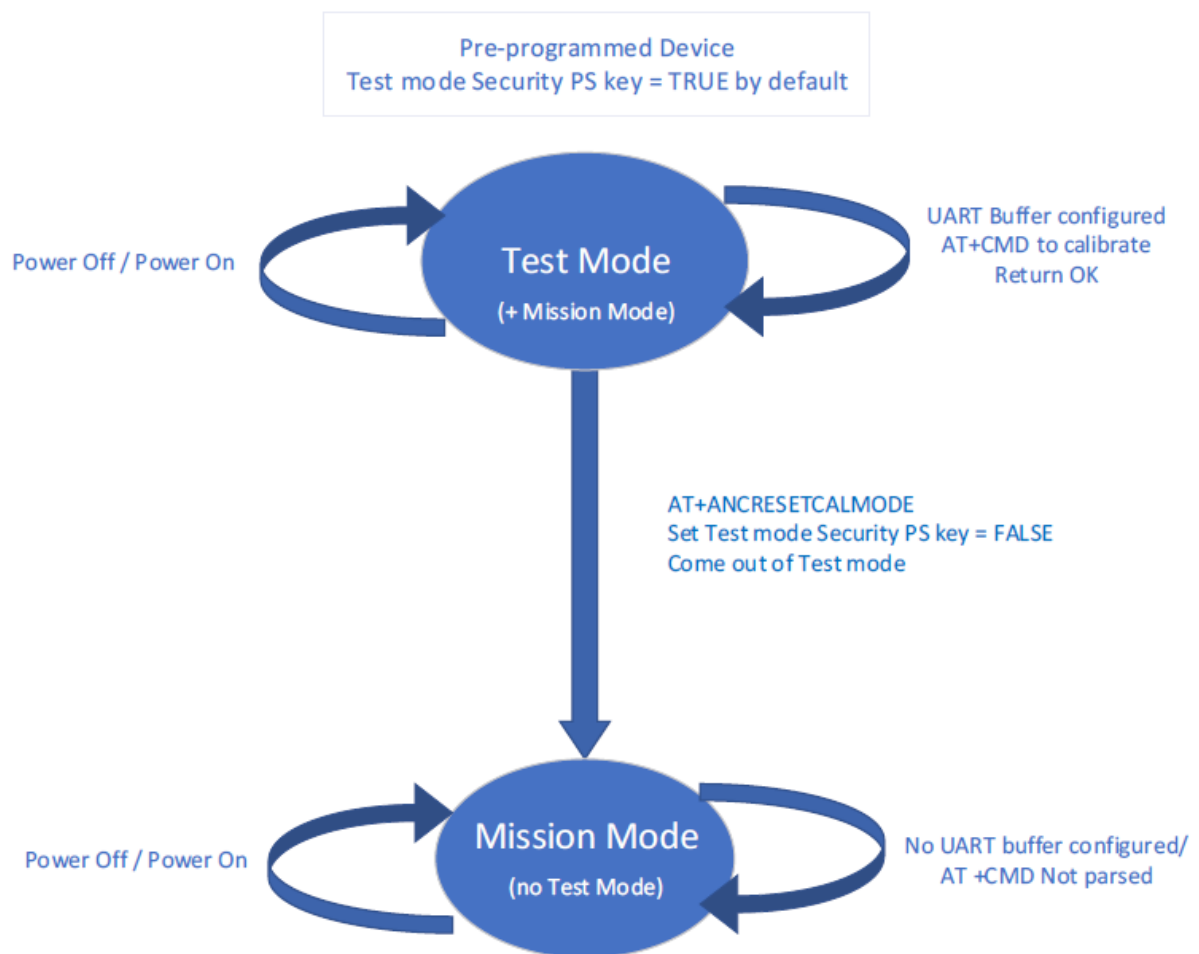


Figure 10-5 Device State

10.3.3.4 Device upgrade

Gain value

ANC gain values are part of the ANC UCID format and are stored under the user PS filesystem.

These keys are persistent across DFU unless explicitly changed by the new DFU image. When a device is upgraded, the ANC gain value is retained in the persistence storage.

Security PS Key

Production-line gain calibration is one-time calibration process. During this process, ensure that the Security PS key is disabled in an upgraded image so that the UART access does not open accidentally.

10.4 Wired calibration

Wired calibration involves calibration using the Test Engine tool, which is shipped as part of Qualcomm® BlueSuite™ with a transport such as USB Debug, SPI, and TRBI200 Interface to communicate directly with the device firmware.

10.4.1 Develop production-line test application

For the production-line, the Test Engine tool is shipped as part of Qualcomm® BlueSuite™ and available as a .DLL shared library.

The tool works on Microsoft Windows only. Test Engine works over various kinds of transport such as USB Debug, SPI, and TRBI200 Interface to communicate directly with the device firmware. The API is provided for the customer to integrate into their final production-line setup. The API allows a host application to control gain values on the device as needed to adjust ANC performance to meet specifications.

The API is flexible and provides the ability to develop a custom test sequence for gain calibration and pass or fail testing. While the example is provided in Python, it is possible to use the `TestEngine.dll` file from another language and use the provided API as guidance for the correct usage and sequence of TestEngine functions.

TestEngine functions

TestEngine tool supports various features including ANC.

NOTE Only functions related to ANC are documented in this section.

For more information on DLL API details, see the `TestEngine help` file located at `{BlueSuite path}\help\TestEngine.chm`.

For Test Engine functions, see *TestEngine Help* documentation available in the *Bluesuite* installation.

Example scripts to show the use of TestEngine API are available at `{BlueSuite path}\ANCTuning\AncProdLine\`.

[Table 10-5](#) lists Test Engine files and their description.

Table 10-5 TestEngine files and their description

File	Description
TestEngine.py	Python wrapper for TestEngine DLL
TestEngine_ANC_API.py	Script with various functions useful for production line calibration
TestEngine_ANC_constants.py	Definition file for setting up ANC-related constants

Table 10-5 TestEngine files and their description (cont.)

File	Description
<code>pull_filter.py</code>	'Pulls' filter data from QCC and writes it to a human-readable text file
<code>push_filter_to_htf.py</code>	'Pushes' a filter data set to the file system in QCC51xx
<code>convert_Filter_to_HTF.py</code>	Converts a filter file in a readable text file format to a .htf filter format that can be loaded into the QCC file system
<code>convert_HTF_to_Filter.py</code>	Converts an HTF file into a filter text file that is easier to interpret
<code>Tuning_demo.py</code>	Example script for a production line tuning demo

10.4.2 Determine calibrated gain parameters

The Python ANC API (file: `TestEngine_ANC_API.py`) includes functions that allow:

- Adjusting gain parameters on the production line.
- Reading parameters from or writing parameters to the HTF file and the file system on a device.

The ANC API provides two sets of functions, Basic functions and Extended functions.

10.4.2.1 Python ANC API example microphone configurations

This section provides example configurations for analog and digital microphones.

Prior to performing gain calibration, the file `TestEngine_ANC_constants.py` needs to be updated with a mic configuration that matches the configuration on the device.

The Microphone instance (`_INST`) can be the value of 0, 1, 2, 3, 4, or None and must match the settings of the UDT. Several presets are provided for frequently used mic configurations such as Stereo hybrid digital mics, Mono hybrid analog mics, Stereo Feed-forward analog mics, and so on.

Example: Feed-forward with digital and feedback modes with analog microphones

```
FFA0_MIC_INST = 1 # mic instance for FFA on ANC0

FFB0_MIC_INST = 1 # mic instance for FFB on ANC0

FFA1_MIC_INST = 2 # mic instance for FFA on ANC1

FFB1_MIC_INST = 2 # mic instance for FFB on ANC1

# mic channel 0=channel A, 1=channel B

FFA0_MIC_CHAN = 0 # mic channel for FFA on ANC0
```

```

FFB0_MIC_CHAN = 1 # mic channel for FFB on ANC0

FFA1_MIC_CHAN = 0 # mic channel for FFA on ANC1

FFB1_MIC_CHAN = 1 # mic channel for FFB on ANC1


# mic type

FFA0_MIC_TYPE = ANALOG_MIC

FFB0_MIC_TYPE = DIGITAL_MIC

FFA1_MIC_TYPE = ANALOG_MIC

FFB1_MIC_TYPE = DIGITAL_MIC

```

10.4.2.2 Python ANC API basic functions

The Basic functions cover basic ANC functionality, such as enabling or disabling ANC operation, setting gains, and converting gains.

The basic ANC API functions along with their descriptions are listed in [Table 10-6](#), [Table 10-7](#), and [Table 10-8](#).

Table 10-6 ANC API functions

Function	Description
connectToDevice	Connect to the device.
getSinkID	Get a handle to sink audio stream. In this case, the speaker connections.
getSourceID	Get a handle to source audio stream. In this case, the microphones.
audioConfigure	Set stream key.
ConfigureStreams	Set gain value via audio Configure.
AncDisable	Disable ANC operation.
AncEnable	Enable ANC operation.
AncGainConvertFromdB	Convert fine and coarse gain to total gain in dB.
AncGainConvertToDb	Convert total gain in dB to fine gain and coarse gain values.
AncSetGain	Set gain value in dB.
SaveGains	Save gain values to GAINS dictionary.

Table 10-7 ANC API: Gain setting functions

Function	Description
AncSetGainFFA0	Set gain in dB on FFA path, left channel.
AncSetGainFFB0	Set gain in dB on FFB path, left channel.

Table 10-7 ANC API: Gain setting functions (cont.)

Function	Description
AncSetGainFB0	Set gain in dB on FB path, left channel.
AncSetGainFFA1	Set gain in dB on FFA path, right channel.
AncSetGainFFB1	Set gain in dB on FFB path, right channel.
AncSetGainFB1	Set gain in dB on FB path, right channel.

Table 10-8 ANC API: Functions for reading and writing parameters

Function	Description
readHTF	Read ANC parameters from the HTF file (generated by QACT).
writeHTF	Write ANC parameters in the local cache to the HTF file.
readAudioKey	Read ANC parameters from the file system on a device.
writeAudioKey	Write ANC parameters to file system on device.
displayAudioKeyData	Print values of parameter local cache to screen.
setParam	Update the value of a parameter in the local cache.
getParam	Retrieve a value of a parameter from the local cache.
updatePsdata	Update the local cache with the last known gain values.
updateGains	Populate the starting gain values from the local cache.
getChipName	Query chip name.
saveANCparams	Write current local cache of ANC block parameters to file system on the device and make a corresponding human-readable text file and .htf file (uses writeAudioKey and writeHTF functions).

10.4.2.3 Python ANC API extended functions

The extended functions may not be useful in factory calibration but can be useful to automate interaction with the ANC block using Python tools.

The functions such as the Connection function, Configuration functions, and Filter read/write functions:

- Extend the available features for ANC configuration.
- Cover more use cases along with improved error-checking depending on the architecture.

Connection function

To connect to the EVB, use the following function:

```
[engine, handle] = ANCCConnectDUT( BluesuitePath, TRANSPORT)
```

This function connects to the device under test using the provided transport.

The sample inputs and outputs are as follows:

Input	Outputs
BluesuitePath: Location for the ADK/BlueSuite files. For example, C:\Program Files (x86)\QTIL\BlueSuite 3.3.16 Transport: TRANSPORT_USDBDG = 256, TRANSPORT_TRB = 128	Engine: Pointer to TestEngine DLL Handle: Handle to device

For example, the following command connects to the EVB using the USB transport:

```
>>> BluesuitePath = r'C:\Program Files (x86)\QTIL\BlueSuite 3.3.16'>>>
engine, handle = ANCCConnectDUT( BluesuitePath, TRANSPORT_USDBDG)
```

Configuration functions

Most of the available Python functions used with the ANC block are formed using a predefined structure:

```
<Command><Feature>_ <Variable>_<Instance><ID>(engine, handle, value[])
```

Function	Description
Command	Describes what needs to be accomplished. For example, enable or disable a feature, set a value. The provided commands are: ■ Enable ■ Disable ■ Set
Feature	Describes the ANC feature with which the command is being used. This includes all the blocks in the ANC hardware and the ANC function itself. Possible values are: ■ ANC ■ DCFilter ■ IIRFilter ■ LPF ■ SMLPF ■ Gain
Variable	This is the characteristic that needs to be modified in each feature. It varies depending on the feature being modified: ■ Shift ■ Coeffs ■ Fine gain ■ dB
Instance	Describes the instance issuing to the command. It can be a particular channel, an individual block, or an architecture. For example: ■ FFa ■ FFb ■ FB ■ ANC ■ Left ■ Right

	■ Hybrid ■ Stereo
ID	Specifies a particular instance. It can be left empty. For example: ■ 0 ■ Left ■ 1 ■ Right ■ <empty>

Python commands and features

A detailed description for each of the commands and features is provided in the `TestEngine_ANC_API.py` file.

The following sample script configures the Left channel with a set of values:

NOTE This is an example script. The values shown in the following example are not intended to be used in the system.

```
>>> import sys, os, time
>>> # Import ANC_API
>>> from TestEngine_ANC_API import *
>>> # Set Bluesuite path
>>> BluesuitePath = r'C:\qtil\ADK_QCC512x_QCC302x_WIN_6.3.0.154\tools\bin'
>>> # Connect to EVB. This example assumes the TRB is being used
>>> engine, handle = ANCCConnectDUT(BluesuitePath, TRANSPORT_TRB)
>>> # Set some variables to be used for configuration
>>> coeffsFeedForward = [0,0,0,0,0,0,0,0,0,0x200,0,0,0,0,0,0]
>>> coeffsFeedback = [0,0,0,0,0,0,0,0,0,0x100,0,0,0,0,0,0]
>>> coeffsEC = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
>>> ShiftLPF0 = 5
>>> ShiftLPF1 = 6
>>> # Disable Left ANC (FFa, FFb, FB) before making changes
>>> DisableANC_ANC0(engine, handle)
>>> # Set LPF cutoff frequency for FFa0
>>> SetLPF_Shift_FFa0(engine, handle, ShiftLPF0, ShiftLPF1)
>>> # Set LPF cutoff frequency for FFb0
>>> SetLPF_Shift_FFbLeft(engine, handle, ShiftLPF0, ShiftLPF1)
>>> # Set coefficients for feedback path
>>> SetIIRFilter_Coeffs_FFa0(engine, handle, coeffsFeedback)
>>> # Set coefficients for feed forward path
>>> SetIIRFilter_Coeffs_FFb0(engine, handle, coeffsFeedForward)
>>> # Set coefficients for EC path
>>> SetIIRFilter_Coeffs_FB0(engine, handle, coeffsEC)
>>> # Enable ANC
>>> EnableANC_ANCLeft(engine, handle, 1)
```


Filter read/write functions

The scripts listed in [Table 10-9](#) provide examples of how to read and write filters from persistent storage.

Table 10-9 Read/write scripts and their usages

Script	Usage
<code>pull_filter.py</code>: The purpose of this script is to pull filter data from QCC and write it to a human-readable text file.	<code>python pull_filter.py X [sampleFilter.txt]</code> Where: ■ <code>X</code> is the ANC filter location to read ■ <code>X=1</code> is the first ANC filter ■ <code>X=2</code> is the second ANC filter ■ <code>X=3</code> is the third ANC filter ■ <code>X=...</code> is the ...th ANC filter ■ <code>X=n</code> is the nth ANC filter If the second argument is present, it denotes the file name where the data will be saved. To execute from within iPython, for example, Anaconda environment: <pre>>>> import sys >>> sys.argv = ['pull_filter.py', '1'] >>> execfile(sys.argv[0])</pre> To print the results of the internal functions on screen, set <code>verboseFlag</code> to 1.
<code>push_filter_to_hw_registers.py</code>: This script allows you to push a filter data set to the hardware registers of the device.	<code>python push_filter_to_hw_registers.py X</code> Where <code>X</code> is a text file with the filter location parameters. NOTE ■ The filter file must be produced from the <code>pull_filter.py</code> script. ■ Data written to the hardware registers will be lost upon power down. To execute from within iPython, for example, Anaconda environment: <pre>>>> import sys >>> sys.argv = ['push_filter_to_hw_registers.py', 'sampleFilter.txt'] >>> execfile(sys.argv[0])</pre> To print the results of the internal functions on screen, set <code>verboseFlag</code> to 1.
<code>push_filter_to_htf.py</code>: This script allows you to push a filter data set to the file system in the device.	<code>python push_filter_to_htf.py X Y ['samplefile.htf']</code> Where: ■ <code>X</code> is a text file with the filter location parameters ■ <code>Y</code> is the ANC filter location to write ■ <code>Y=1</code> is the first ANC filter ■ <code>Y=2</code> is the second ANC filter

Table 10-9 Read/write scripts and their usages (cont.)

Script	Usage
	■ Y=3 is the third ANC filter ■ Y= is the ...th ANC filter ■ Y= 10 is the tenth ANC filter If the third argument is present, it denotes the file name where the data will be saved, when the data is in HTF format. To execute from within iPython, for example, Anaconda environment: <pre>>>> import sys >>> sys.argv = ['push_filter_to_htf.py', 'sampleFilter.txt', '1', 'samplefile.htf'] >>> execfile(sys.argv[0])</pre> To print the results of the internal functions on screen, set <code>verboseFlag</code> to 1.

10.4.2.4 Python ANC API variables

Table 10-10 lists variables that are useful in deploying ANC to production.

Table 10-10 Python ANC API variables

Variable	Description	Notes
psdata	The cache of ANC parameters as read from the device.	371 16-bit elements
GAINS	The cache of last gain values written to the device.	dict with gains including shift, fine gain, and total dB
handle	Handle to the device, usually enumerates as 1	-
engine	TestEngine.dll interface	-
BluesuitePath	Path to location of TestEngine.dll (and dependencies)	Can point to a BlueSuite installation or ADK tools/bin folder

10.4.3 Gain parameters and calculation

ANC parameters are generated by the ANC filter designer and saved in an HTF file. These parameters are stored on the device in audio persistence storage. During production-line calibration, the parameters are read and written directly to persistence.

10.4.3.1 ANC block gain parameters

Table 10-11 lists the gains inside the ANC block and how they are related to stream keys as used by the application.

Table 10-11 ANC block gain parameters

Name	Stream key (apps P0)	ANC Parameter (audio SS)
ANC0 FFA fine gain	STREAM_ANC_FFA_GAIN	OFFSET_ANC_FF_A_GAIN_L
ANC0 FFA coarse gain	STREAM_ANC_FFA_GAIN_SHIFT	OFFSET_ANC_FF_A_SHIFT_L
ANC0 FFB fine gain	STREAM_ANC_FFB_GAIN	OFFSET_ANC_FF_B_GAIN_L
ANC0 FFB coarse gain	STREAM_ANC_FFB_GAIN_SHIFT	OFFSET_ANC_FF_B_SHIFT_L
ANC0 FB fine gain	STREAM_ANC_FB_GAIN	OFFSET_ANC_FB_GAIN_L
ANC0 FB coarse gain	STREAM_ANC_FB_GAIN_SHIFT	OFFSET_ANC_FB_SHIFT_L
ANC1 FFA fine gain	STREAM_ANC_FFA_GAIN	OFFSET_ANC_FF_A_GAIN_R
ANC1 FFA coarse gain	STREAM_ANC_FFA_GAIN_SHIFT	OFFSET_ANC_FF_A_SHIFT_R
ANC1 FFB fine gain	STREAM_ANC_FFB_GAIN	OFFSET_ANC_FF_B_GAIN_R
ANC1 FFB coarse gain	STREAM_ANC_FFB_GAIN_SHIFT	OFFSET_ANC_FF_B_SHIFT_R
ANC1 FB fine gain	STREAM_ANC_FB_GAIN	OFFSET_ANC_FB_GAIN_R
ANC1 FB coarse gain	STREAM_ANC_FB_GAIN_SHIFT	OFFSET_ANC_FB_SHIFT_R

10.4.3.2 Write calibrated gain parameters to device persistence

The ANC API gain functions sets the gain immediately on the device by writing to hardware. The `GAINS` variable tracks the last written gain.

After gains are finalized:

- Call `updatePsdata` to merge the gain values from `GAINS` into `psdata`.
- Call `writeAudioKey` to write `psdata` into the persistence of the device.

Figure 10-6 shows the writing calibrated gain parameters and their functions.

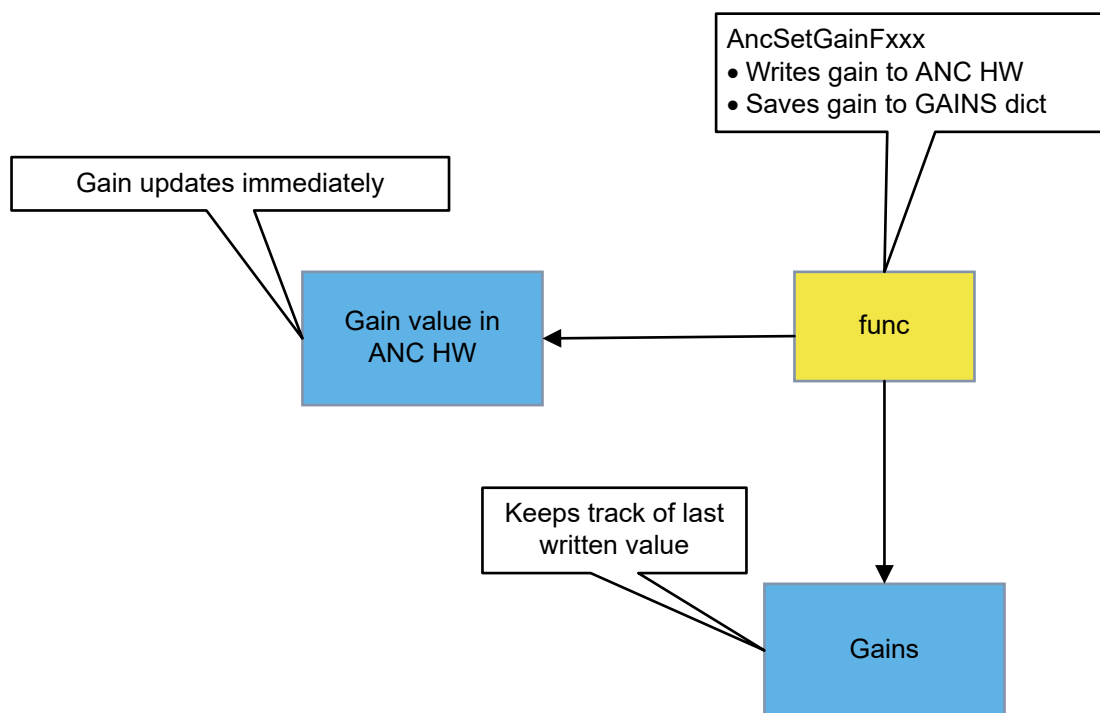


Figure 10-6 Writing calibrated gain parameters

Read and write to parameter storage

In the audio PS Key, parameters are stored on the device. The `readAudioKey` and `writeAudioKey` functions copy ANC parameter data to and from the device.

`psdata` is a local cache of ANC parameters in Python scripts. Whenever a function is called to change a parameter in the ANC block, the local cache `psdata` is also updated with the corresponding change. After a set of ANC parameters is found that results in satisfactory ANC performance, `writeAudioKey` or `saveANCparams` can write the parameters to the chip.

Figure 10-7 shows how the reading and writing to parameter storage function is carried out.

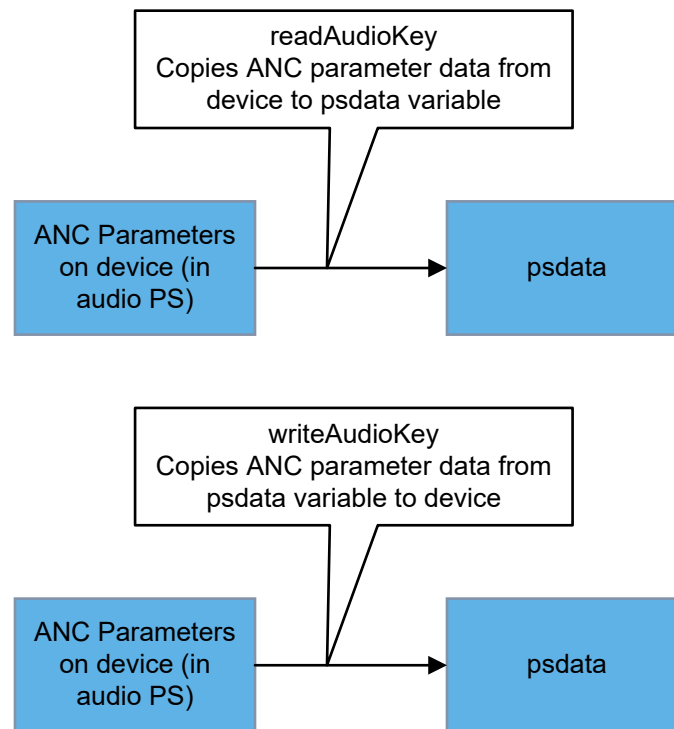


Figure 10-7 Reading from and writing to parameter storage

Read and write to HTF file

Parameters can be stored in an HTF file. HTF files are compatible with QACT and Qualcomm MDE.

The `readHTF` and `writeHTF` functions of the ANC API copy ANC parameter data to and from the HTF file. `psdata` is a local cache of ANC parameters in the Python scripts.

Figure 10-8 shows how the reading and writing to HTF file function is carried out.

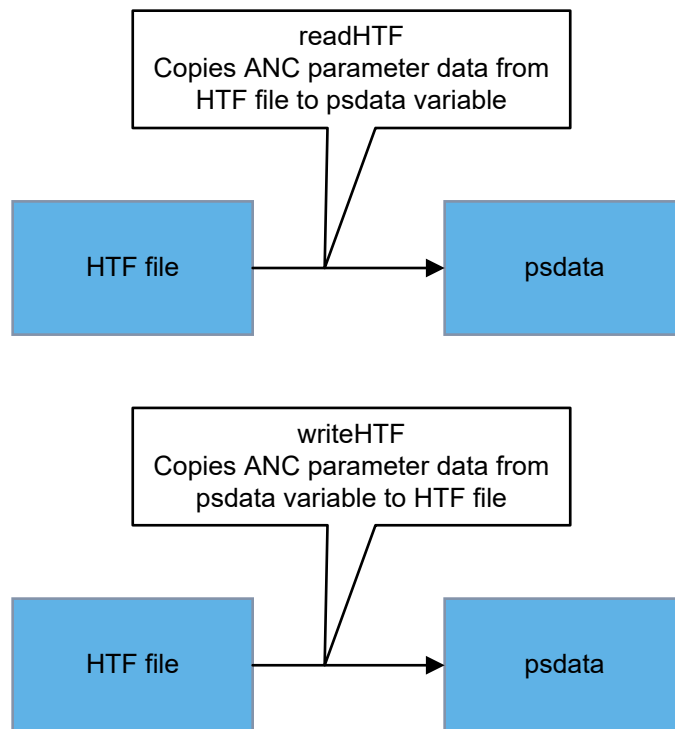


Figure 10-8 Reading from and writing to the HTF file

Example: Write gain parameters

The Python code example in this section shows the flow for using the production-line API. For more information, see the comments describing each section of the code.

```
# current file path of latest version of BlueSuite

BluesuitePath = r'C:\Program Files (x86)\QTIL\BlueSuite 3.2.1'

# instantiate test engine object

engine = TestEngine(BluesuitePath)

# get handle to device over specified transport

handle = connectToDevice(engine, TRANSPORT_USBDDBG)

# query chip name

getChipName(engine, handle)

# read ANC parameters from device into psdata cache

readAudioKey(engine, handle, ANC_PSKEY_ACTIVE)
```

```
# update local gains cache with values from psdata

updateGains()

# display parameters in psdata cache to screen

displayAudioKeyData()

'''

place test code here to adjust gain values

example calls to gain adjustment API shown below

'''

AncSetGainFFA0(engine, handle, 5

AncSetGainFFA1(engine, handle, 12)

#AncSetGainFB0(engine, handle, -3)

#AncSetGainFFA1(engine, handle, -15)

#AncSetGainFFB1(engine, handle, 22.5)

#AncSetGainFB1(engine, handle, 36)

# write data from gains cache into psdata cache

updatePsdata()

# write psdata cache to device

writeAudioKey(engine, handle, ANC_PSKEY_ACTIVE)

# save HTF file with final tuned parameters(optional)

writeHTF()

# close connection to device

engine.closeTestEngine(handle)
```

Configure other ANC parameters using TestEngine

TestEngine API can also be used to configure other ANC-related parameters such as filter coefficients, LPF and DC block. For more information, see the `python script` file.

11 Adaptive ANC tuning use cases

The Qualcomm ADK includes various audio capabilities or functionalities, for example Earbud Fit Test and DSP-based leak-through that are supported on the chip. These capabilities can be tuned based on the design requirements.

11.1 Earbud Fit Test feature

The Earbud Fit Test feature (EFT) provides a user-triggered test to determine whether the earbuds fit properly. A music file stored in the filesystem is played synchronously on both earbuds and the assessment of the fit quality is made on each side.

The audio graph includes two capabilities that perform the following functions:

1. Adaptive ANC (AANC), which monitors the environment to help reduce invalid outcomes.
2. EFT, which evaluates the quality of the fit in each earbud and sends a message to the application when a good fit has been detected.

It is particularly important to tune the threshold to determine a good fit (see [Tune the Earbud Fit Test capability](#)), as this varies depending on the earbud acoustics.

11.1.1 Build and configure the Earbud Fit Test capability

Before you begin, ensure that the ANC and Adaptive ANC features are enabled, as described in [Enable ANC features in the Earbuds application](#). Add the Earbud Fit Test (EFT) definitions to the project in Qualcomm MDE.

Enable the EFT capability in the Earbuds project

To enable the EFT capability, add the `ENABLE_EARBUD_FIT_TEST` definitions to the Earbud project properties:

1. Go to **MDE Projects > General > DEFS**.
2. Update the Feature license for the EFT `subsys7_config3.htf`.
3. Add prompt file named `fit_test.sbc` to the `ro_fs` folder.

NOTE The file name is case sensitive and the application searches specifically for this file name only. If any other prompt file is used for testing, rename the file.

4. Check that the `download_earbud_fit_test.edkcs` file is present in the `ro_fs` folder. If not, copy it from the `...\audio\<chip variant>\kalimba_ROM_xxxx\kymera\prebuilt_dkcs\streplus_rom_release\...` folder, and then add it manually to the `ro_fs` folder.
5. Update the `anc_tuning_config.htf` (UCID=63), `aanc_parameters.htf` (UCID = 63), and `eft_tuning_config.htf` (UCID = 0) files with tuned parameters for fit test. These files are in the `tps_cfg` folder.

Debug with Pydbg

Table 11-1 lists the Pydbg commands for EFT.

Table 11-1 Pydbg test commands for Earbuds Fit Test

Test Interfaces	Usage
<code>apps1.fw.call.EarbudTest_StartFitTest()</code>	Starts the EFT. The prompts get played in a loop for predefined number of times.
<code>apps1.fw.call.EarbudTest_StopFitTest()</code>	Stops/Aborts the ongoing EFT.
<code>apps1.fw.call.EarbudTest_GetLocalDeviceFitTestStatus()</code>	Gives the information about the results of EFT of the local device. Return value: 0: Bad, 1: Good, 2: Error. Result shown by this API is valid only after the EFT is complete. NOTE Checking the API in between the test results in garbage value.
<code>apps1.fw.call.EarbudTest_GetRemoteDeviceFitTestStatus()</code>	Gives the information about the results of the EFT of the remote device. Return value: 0: Bad, 1: Good, 2: Error. Result shown by this API is valid only after the EFT is complete. NOTE Checking the API in between the test results in garbage value.
<code>apps1.fw.var.fit_test_task_data</code>	Strictly for debugging purpose. Shows the internal state of <code>fit_test</code> task. Information about the internal state machine for EFT can be found using the same.

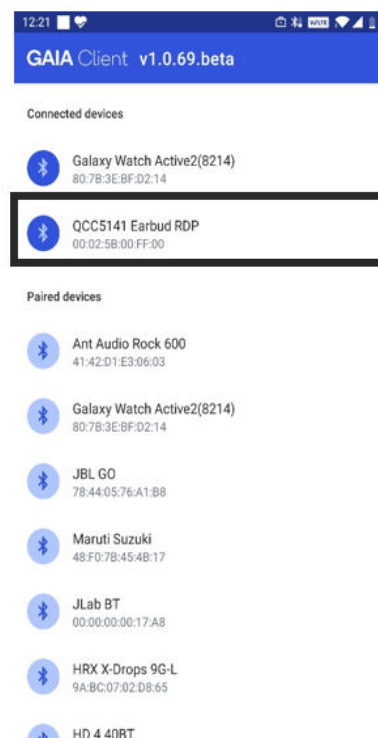
Table 11-1 Pydbg test commands for Earbuds Fit Test (cont.)

Test Interfaces	Usage
<code>apps1.fw.call.EarbudTest_StartFitTestTuning()</code>	Starts the fit test chain in the tuning mode. Fit test chain is kept alive using the fit-test prompt in a repetitive manner. The Pydbg call for <code>StopFitTestTuning()</code> is required for stopping the tuning chain.
<code>apps1.fw.call.EarbudTest_StopFitTestTuning()</code>	Stops the EFT tuning chain if already running.

Perform EFT using GAIA

To perform the EFT using the GAIA application:

1. On your mobile device, start the GAIA Application for earbuds.
2. Select the required earbuds.

**Figure 11-1 available earbuds in the GAIA application**

3. When the earbud is connected, tap **Settings**.

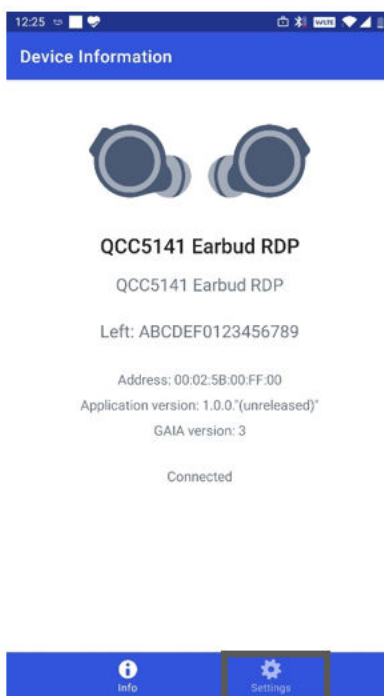


Figure 11-2 Earbud settings

4. Start the EFT by tapping **GAIA Application > Earbud Fit Test > Start Test**.
A test prompt is played during the test run.
5. After the test is complete, the results (good fit, bad fit, or inconclusive) are shown for each earbud.

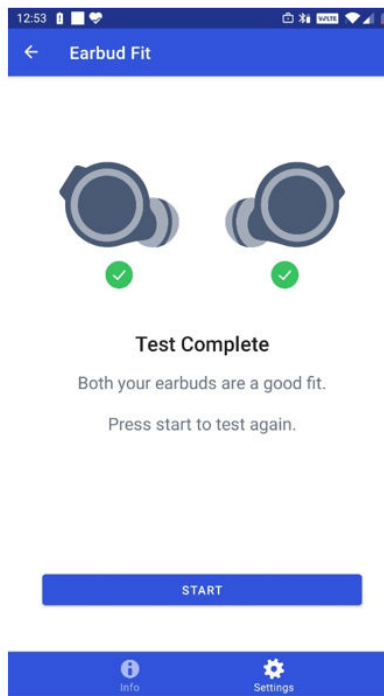


Figure 11-3 Earbud test results

11.1.2 Source file design for Earbuds Fit Test

The source file design involves selecting acoustic properties to identify the right type of music to play during wear status detection, and choosing the recommended encoding for the source file.

Acoustic properties

Selecting the correct type of music to play during wear status detection is extremely important.

The following requirements must be met:

- Frequency of 60 Hz to 140 Hz.
- Music without vocals.
- Homogenous content throughout the whole clip. Having different sections with voice, percussion, and other instruments does not produce good results.
- Very few silence segments. A silence segment should not be longer than 20 ms.

Figure 11-4 shows a clip that meets following requirements:

- Low frequencies are consistent across the whole clip.
- Sound content is consistent across the clip. Amplitude is constant.
- Silence segments are not present.

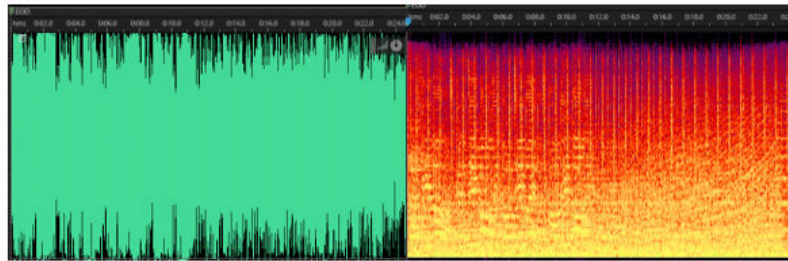


Figure 11-4 Example of a good source file for wear status tuning

Figure 11-5 shows a clip that fails to meet the following requirements:

- Sections of the clip are missing low frequencies (60 Hz to 120 Hz). Only the middle section has these frequencies.
- Non-homogenous clip. Sections are very different with some including only voice, other sections including only strings, and the rest of the sections are silence segments. Amplitude also varies greatly across the clip.
- Silence segments are present.

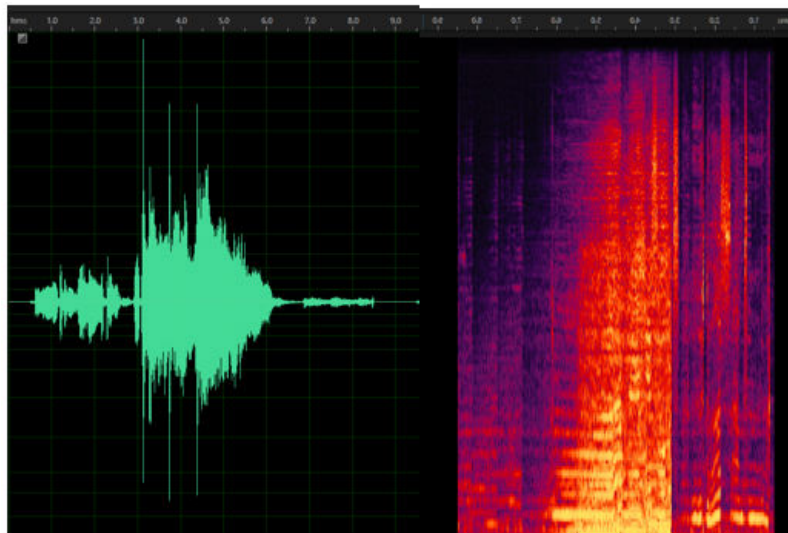


Figure 11-5 Example of a bad source file for wear status tuning

When tuning with the right music, the Power Ratio plot should have low variability for a fixed fit, as shown in Figure 11-6.

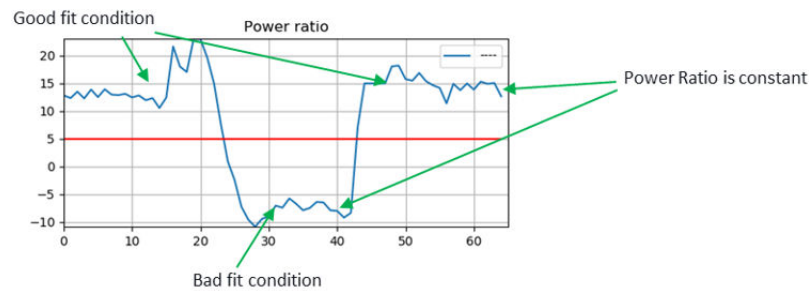


Figure 11-6 Example of the Power Ratio for a good source file

When incorrect file is used, the Power Ratio plot varies too much for a fixed fit and causes false triggers, as shown in Figure 11-7.

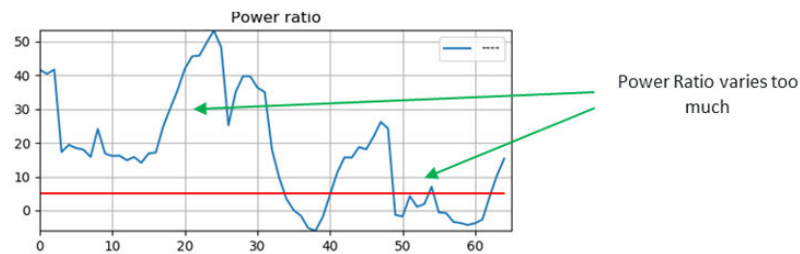


Figure 11-7 Example of the Power Ratio for a bad source file

Recommended encoding

The file must be SBC encoded and included in the device filesystem.

The recommended encoding parameters are:

- 48 kHz sample rate
- Mono channel
- 16 blocks per frame
- 8 sub-bands
- Bitpool: 53
- Duration between 5 and 10 seconds

A lower sample rate can be used to reduce the file size. However, 48 kHz is recommended for high quality playback.

11.1.3 Tune the wear status graph

A wear status graph comprises both the Adaptive ANC capability for environment detection and the EFT capability for evaluating the quality of the fit.

11.1.3.1 Environment monitoring using the Adaptive ANC capability

The Adaptive ANC capability is configured to monitor the environmental conditions, including the ambient noise level, touch sensitivity of the Earbud during the test, and any non-stationary noise detected during the test. Each condition has an associated message that is sent to the application to if the wear status environment is not suitable for the evaluation.

Tune the ambient noise level

The AANC parameters involved in monitoring the ambient noise level are *Hold Time: SPL Event* and *SPL Event Threshold*. The Hold Time: SPL Event controls the amount of time the SPL event must be active before a message is sent. The SPL Event Threshold controls the threshold above which this event triggers.

Figure 11-8 shows the ambient level condition parameters in QACT.

— Hold Time: SPL Event	0x00080000	0.500000 s
— SPL Event Threshold	0xFBA00000	-70.0000 dB

Figure 11-8 Ambient level condition parameters

To tune the threshold:

1. Place the earbud in a quiet environment and record the SPL level on the external microphone.
2. Observe how the SPL changes with increased ambient noise.
3. Set the threshold so that it distinguishes between a quiet and noisy environment.

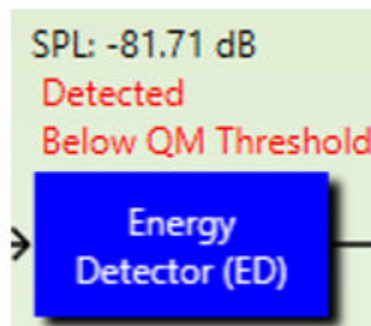


Figure 11-9 Threshold observations in QACT

The `aanclogger` capture shows the speed with which the event is sent, as shown in [Figure 11-10](#).

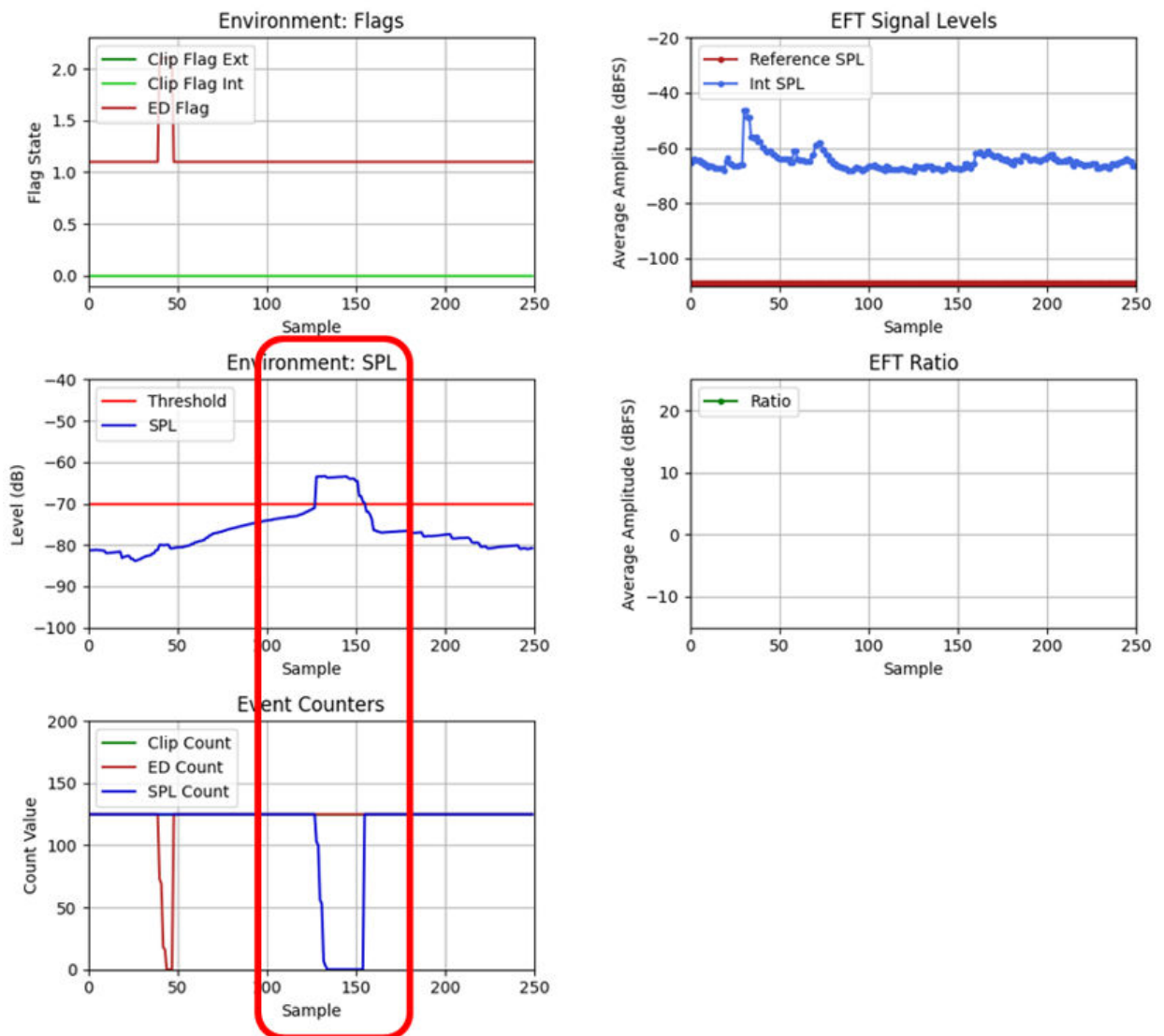


Figure 11-10 Threshold observation using aanclogger

When the SPL goes above the threshold (approximately 130 samples in the capture), the counter value starts decrementing. When it reaches 0, the capability sends a message.

Configure the touch sensitivity of earbuds

The AANC parameters that control the touch sensitivity of earbuds are *Ext Mic Clipping Duration*, *Int Mic Clipping Duration*, and *Hold Time*, which are described in [Table 11-2](#).

Table 11-2 AANC parameters to configure touch sensitivity of Earbuds

Parameter	Description
Ext Mic Clipping Duration	Controls how long a clip detection on the external microphone is held for.
Int Mic Clipping Duration	Controls how long a clip detection on the internal microphone is held for.
Hold Time	Controls the amount of time the clip event must be active before a message is sent.

[Figure 11-11](#) shows the clipping condition parameters in QACT.

Ext Mic Clip Duration	0x00080000	0.500000	s
Int Mic Clip Duration	0x00080000	0.500000	s
Hold Time: Unchanged Clip Event	0x00080000	0.500000	s

Figure 11-11 Clipping condition parameters

The most robust tuning is to have these values matched so that any clipping detected by the capability is communicated as a message. If this needs to be adjusted, consider reducing the clip duration on the internal and external microphones to make it less sensitive.

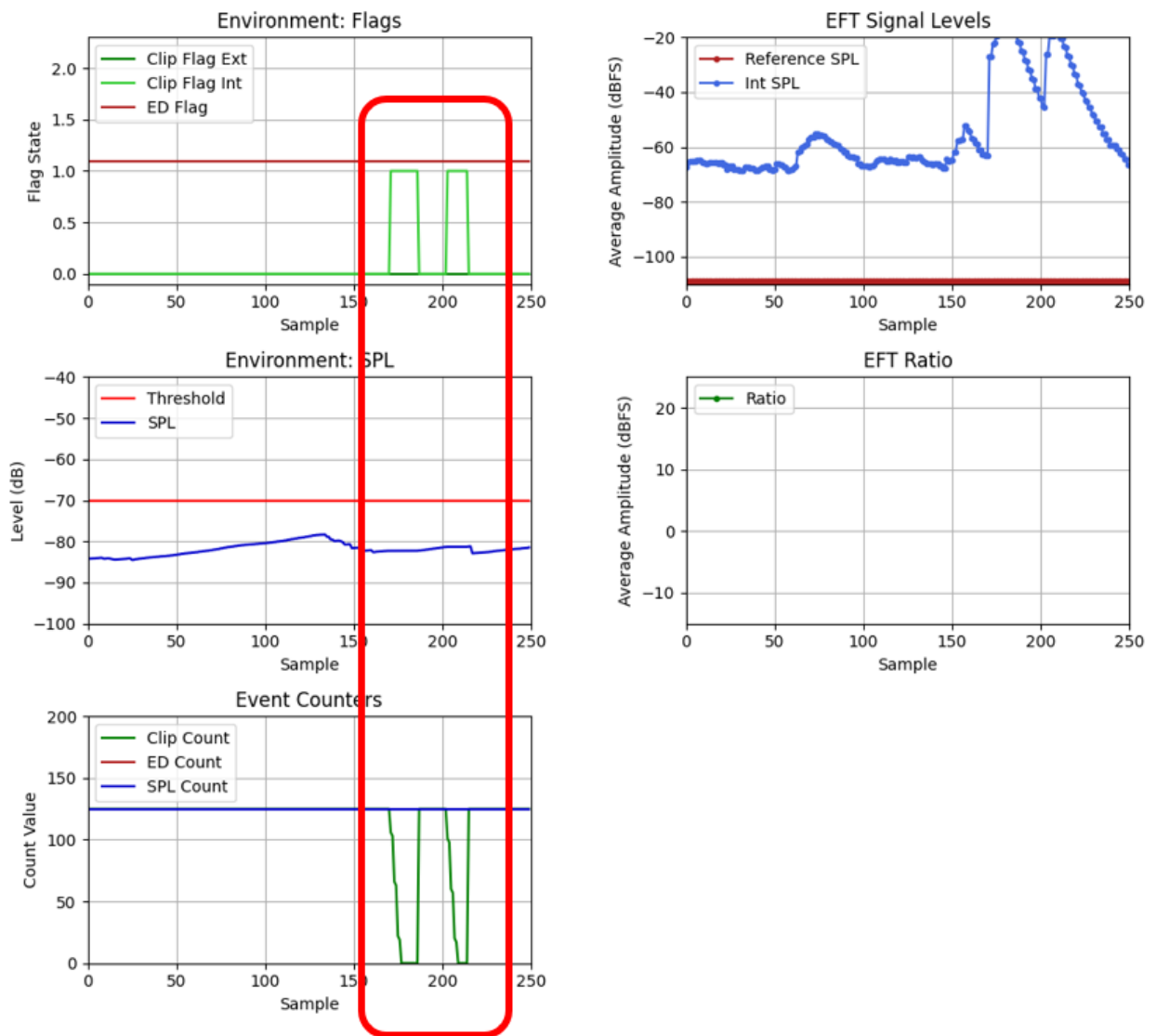


Figure 11-12 Clipping observations using aanclogger

NOTE In the `aanclogger` capture, the clipping detection quickly triggers the counter, and a message is sent when the counter is 0.

Detect non-stationary noise during Earbud Fit Test

The parameter used for detecting non-stationary noise is *Hold Time: Unchanged ED Event*, which controls the amount of time the ED event must be active before a message is sent.

To observe this parameter in QACT, review the `detected` flag on the external ED block (see [Tune the ambient noise level](#)). This flag should be set to correspond to very short bursts of noise, as these might interfere with the test.

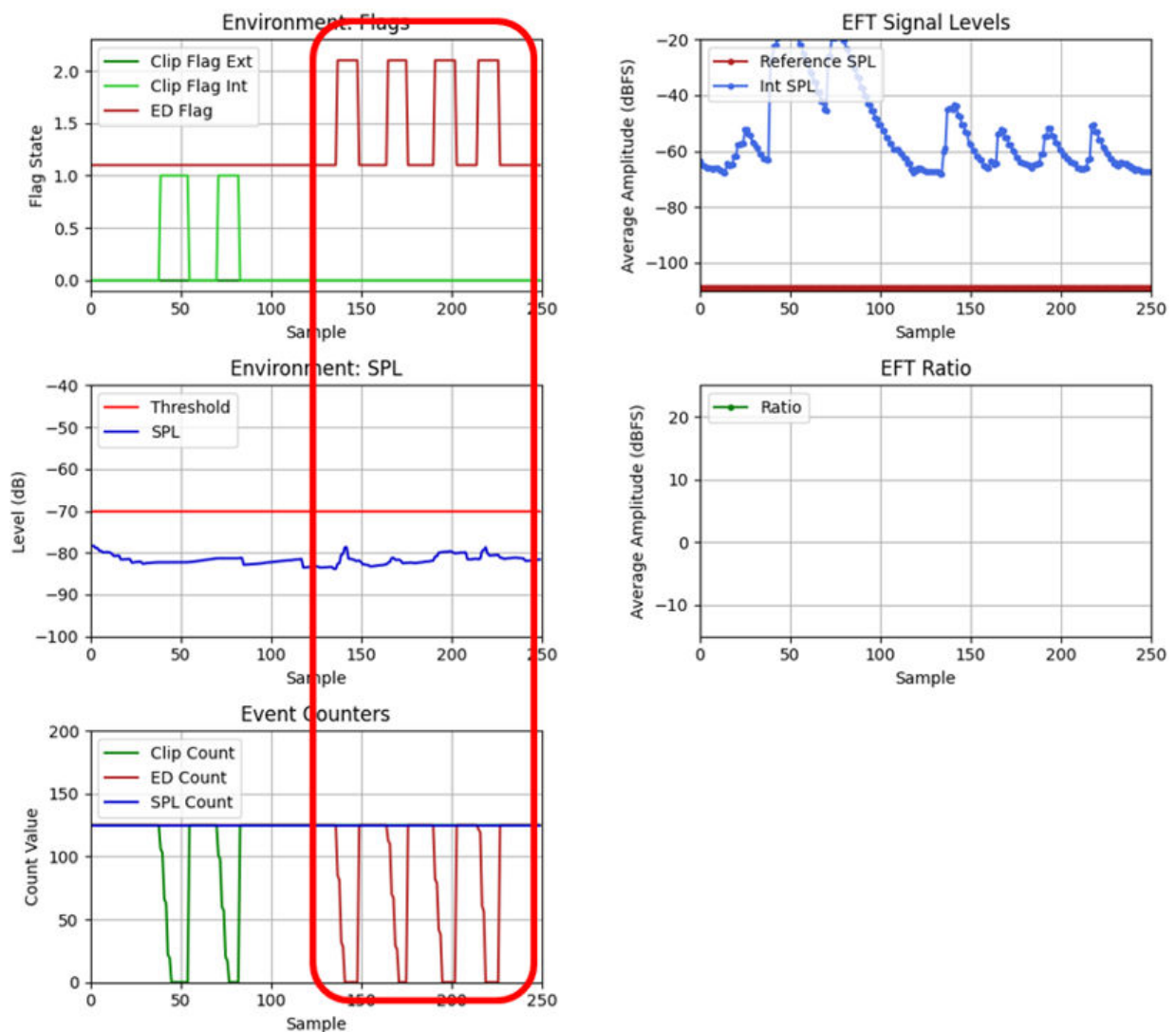


Figure 11-13 ED observations using aanclogger

NOTE The event duration is much longer than the time taken to send the message (for the counter to decrement to 0).

11.1.3.2 Tune the Earbud Fit Test capability

The EFT tuning process involves setting a threshold that can differentiate between a good or bad fit. When a good fit is attained, the fit detector plays a predetermined music selection and evaluates the results in the internal microphone.

Threshold tuning

To tune the EFT capability, use the `Fit Quality Threshold` parameter.

Fit Quality Threshold 0x04000000 4.000000

Figure 11-14 Earbud fit test threshold parameter

Play the selected music in a loop and check the measurements of the Power Ratio statistic for different cases. Based on these values, a threshold is selected such that its value is equidistant to the recorded statistics.

- Good fit power ratio value: While playing the selected music, set the earbud in the ear with a good fit. As an alternative, block the ear-tip with a finger. When the power ratio plot has converged and is steady, record its value.

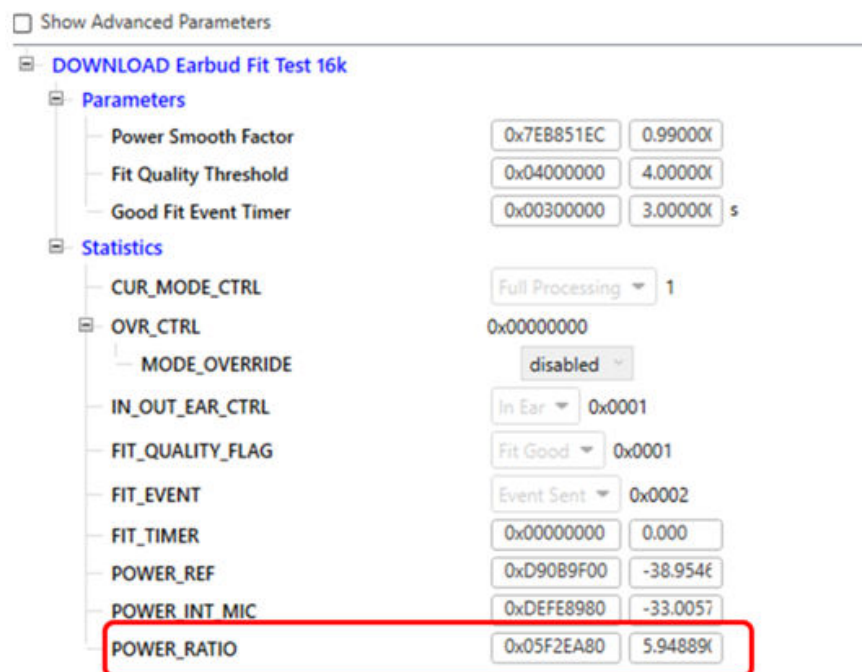


Figure 11-15 QACT example showing a good fit power ratio

- **Bad fit power ratio value** While playing the selected music, keep the earbud open, without any blocks, for example keep it on the desk. When the power ratio plot has converged and is steady, record its value.

After obtaining both power ratio values, find the middle value and use it as the threshold.

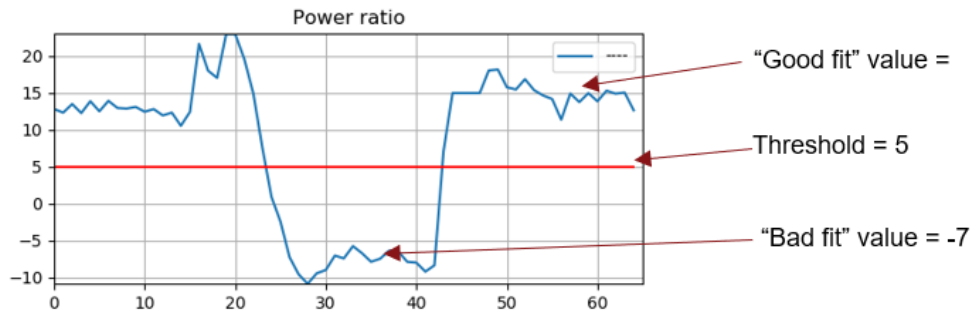


Figure 11-16 Example power ratio plot

Event timer tuning

When the power ratio crosses the threshold for a good fit, a counter is started. This counter determines how long the power ratio must be in the good fit condition for the test to pass. The Good

Fit Event Timer parameter in QACT controls this counter. The counter value appears in QACT under the FIT_TIMER statistic.

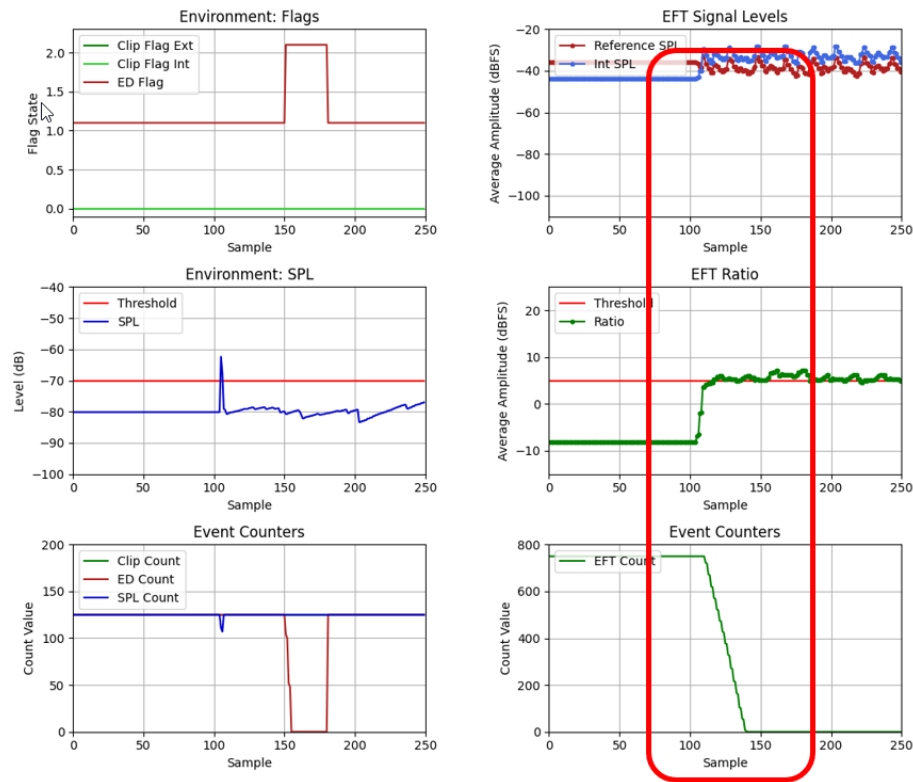


Figure 11-17 EFT example using aanclogger

NOTE In the `aanclogger` capture, the ratio crosses the threshold and causes the counter to decrement. The count value is in frames, in this case decrementing from 750 to 0 in three seconds.

11.2 Wind Noise Detect feature

The Wind Noise Detect (WND) feature supports wind noise detection and mitigation in ANC use cases, specifically in Adaptive ANC/Leak-through and Static ANC/Leak-through modes, with Adaptive ANC build configurations.

Wind noises must be accurately detected and feed-forward (FF) gain is adjusted (overriding existing gain) to remove the wind noise. To do that, ANC FF and Diversity microphones are deployed. This feature supports Stage 1 and Stage 2 wind detection mechanism based on the microphone availability and settings. Along with wind detection algorithm, this feature also provides power-level based wind intensity indicator that determines the best mitigation. For a product with non-shared ANC FF and Diversity microphones, stage 2 wind detection is possible. The feature is supported on QCC517x/QCC518x and QCC307x/QCC308x/QCC309x chips.

11.2.1 Wind Noise Detect (WND) capability modes

The Wind Noise Detect capability detects the presence of wind turbulence in the external microphones, so that appropriate actions can be taken to mitigate its effects.

The capability can perform wind detection in two modes:

- 1-Mic mode: The capability processes audio data from external microphone.
- 2-Mic mode: The capability processes audio data from external microphone and diversity microphone (can be shared voice mic).

The 2-Mic mode is more robust as it offers better granularity to detect gusts of wind and is easier to tune at the expense of increased power consumption. Both microphones must be synchronized for 2-Mic mode.

Both the algorithms return a flag that indicates the presence of wind in the current frame and a state that indicates the intensity of the wind present around the external microphone.

11.2.2 Wind noise mitigation

Wind Noise Detect capability sends an unsolicited message to application when algorithms detect presence of wind in the environment. Application then trigger wind noise mitigation by setting AHM sysmode to `windy`. This leads to reduction of gain in FF/FB path as per tuned parameters in AHM.

When intensity categorization is enabled in the build, WND capability sends unsolicited messages whenever there is change in intensity. Application triggers wind noise mitigation by reading relevant parameters from WND capability and overriding them in AHM capability before updating it's sysmode.

11.2.3 Build and configure the Wind Noise Detection capability

Before you begin, ensure that the ANC and Adaptive ANC features are enabled, as described in [Enable ANC features in the Earbuds application](#). To build and configure the Wind Noise Detection (WND) capability, add the required definitions to the project in Qualcomm MDE.

Enable the WND capability in the Earbuds project

The WND capability works with Adaptive ANC Feature license (see `subsys7_config3.htf`).

To enable the WND capability:

1. Go to **MDE Projects > General > DEFS**, and then add the `ENABLE_WIND_DETECT` definitions in the earbud project properties.
2. Go to **MDE Projects > General > DEFS**, and then add the `ENABLE_WIND_INTENSITY_DETECT` definition in the earbud project properties to enable wind intensity categorization and mitigate wind noise according to the intensity.
3. Check that the `download_wind_noise_detect.edkcs` file is present in the `ro_fs` folder. If it is not present, add it manually from the `... \audio \<chip variant> \kalimba_ROM_XXXX \kymera \prebuilt_dkcs \maor_rom_release \...` folder.

The <ADK path>\src\domains\audio\kymera\kymera_config.h has the following default configurations for the WND capability:

```
#define appConfigWindDetect2MicSupported() (TRUE)
#define appConfigWindDetectANCFFMic() (appConfigAncFeedForwardMic())
#define appConfigWindDetectDiversityMic() (appConfigMicVoice())
```

NOTE If `appConfigWindDetect2MicSupported()` is set as `TRUE`, make sure that `appConfigWindDetectANCFFMic()` and `appConfigWindDetectDiversityMic()` are not shared.

Debug with Pydbg

Table 11-3 describes the Pydbg commands for testing the WND capability.

Table 11-3 Pydbg WND test commands

Test interface	Simulates...										
apps1.fw.call. EarbudTest_SetStageOneWindDetect ()	Stage one wind detect message										
apps1.fw.call.EarbudTest_ SetStageTwoWindDetect ()	Stage two wind detect message										
apps1.fw.call.EarbudTest_ SetStageOneWindReleased ()	Stage one wind release message										
apps1.fw.call.EarbudTest_ SetStageTwoWindReleased ()	Stage two wind release message										
apps1.fw.call.EarbudTest_SetStageOneWindDetectWith Intensity(intensity_info)	Simulates the Stage one wind detect message from DSP capability with intensity info.										
<table><tr><th>Intensity info</th><th>Value</th></tr><tr><td>Intenisty_none</td><td>0</td></tr><tr><td>Intenisty_low</td><td>1</td></tr><tr><td>Intenisty_medium</td><td>2</td></tr><tr><td>Intenisty_high</td><td>3</td></tr></table>		Intensity info	Value	Intenisty_none	0	Intenisty_low	1	Intenisty_medium	2	Intenisty_high	3
Intensity info		Value									
Intenisty_none		0									
Intenisty_low		1									
Intenisty_medium		2									
Intenisty_high	3										
apps1.fw.call.EarbudTest_SetStageTwoWindDetectWith Intensity(intensity_info)	Simulates the Stage two wind detect message from DSP capability with intensity info.										

Wind Noise Detection using the GAIA Application

To perform Wind noise detection:

1. Start the GAIA Application for earbuds.
2. Select the earbuds to connect to, and then go to **Audio Curation > Demo Mode**.

The Demo mode includes the following options:

- User configuration options to Enable/Disable Wind detection.
- Indication if Wind noise reduction is active. The configured wind mitigation gain is activated in the device and displayed on GAIA app as feed-forward gain.

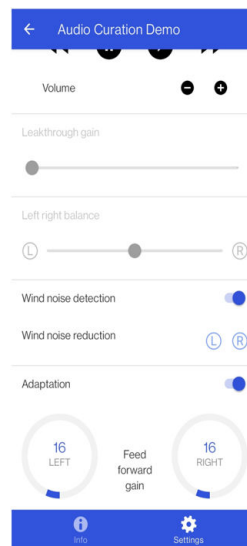


Figure 11-18 GAIA Application: Audio curation Demo mode

11.2.4 Tune the Wind Noise Detection feature

Tuning the Wind Noise Detection (WND) capability involves tuning wind detection for 1-mic and 2-mic modes, tuning the wind intensity, and configuring the state machine behavior for sending messages about wind detection.

11.2.4.1 Wind detection tuning

Wind detection must be tuned separately for 1-mic and 2-mic modes. Each mode has dedicated parameters to tune. To tune wind detection in each mode, there are three parameters available. Two of these parameters control the smoothing filter. The third parameter determines the threshold value, above which the wind is detected.

[Table 11-4](#) lists the wind detection tuning parameters.

Table 11-4 Tunable parameters

Mode	Parameter	Unit	Range	Description
1-Mic	1MIC_FILTER_ATTACK	sec	0 to 20	1 mic filter attack time duration
1-Mic	1MIC_FILTER_DECAY	sec	0 to 20	1 mic filter decay time duration

Table 11-4 Tunable parameters (cont.)

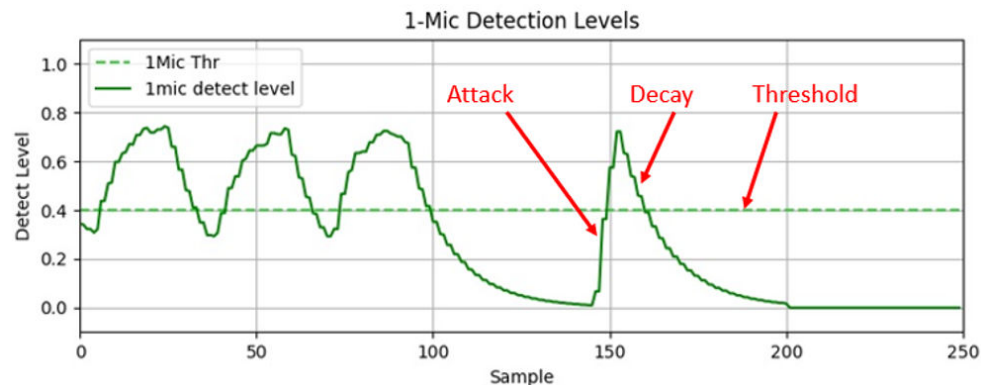
Mode	Parameter	Unit	Range	Description
1-Mic	1MIC_THRESHOLD	-	0 to 1	1 mic threshold for wind detection
2-Mic	2MIC_FILTER_ATTACK	sec	0 to 20	2 mic filter attack time duration
2-Mic	2MIC_FILTER_DECAY	sec	0 to 20	2 mic filter decay time duration
2-Mic	2MIC_THRESHOLD	-	0 to 1	2 mic threshold for wind detection

The smoothing filter is configured using:

- `xMIC_FILTER_ATTACK`: Attack time constant that indicates how fast the envelope tracks values higher than the current estimate.
- `xMIC_FILTER_DECAY`: Decay time constant that indicates how fast this estimate decays for lower values.

The `xMIC_THRESHOLD` parameter determines the threshold value, above which the wind is detected.

Figure 11-19 shows the graphical view.

**Figure 11-19 1-Mic detection levels**

11.2.4.2 Wind intensity tuning

The wind intensity algorithm calculates the power of the signal in the external microphone and displays it in dBFS (full scale). There are seven tunable parameters for detecting wind intensity. These parameters are common for both 1-mic and 2-mic modes.

Table 11-5 lists the parameters used for tuning the wind intensity.

Table 11-5 Wind intensity parameters

Mode	Parameter	Unit	Range	Description
1-Mic/2-Mic	POWER_THRESHOLD	-	0 to 1	Power threshold for running any WND algorithm.
1-Mic/2-Mic	POWER_ENVELOPE_TIME	sec	0 to 20	Power measurement envelope smoothing time duration.
1-Mic/2-Mic	POWER_ATTACK_TIME	sec	0 to 20	Power measurement attack time duration.

Table 11-5 Wind intensity parameters (cont.)

Mode	Parameter	Unit	Range	Description
1-Mic/2-Mic	MED_WIND_THRESHOLD	dBFS	-120 to 0	Used in low to medium intensity transition, below which intensity is 'Low' and above is 'Medium'.
1-Mic/2-Mic	HIGH_WIND_THRESHOLD	dBFS	-120 to 0	Used in medium to high intensity transition, below which intensity is 'Medium' and above is 'High'.
1-Mic/2-Mic	MED_REL_WIND_THRESHOLD	dBFS	-120 to 0	Used in medium to low intensity transition, below which intensity is 'Low' and above is 'Medium'.
1-Mic/2-Mic	HIGH_REL_WIND_THRESHOLD	dBFS	-120 to 0	Used in high to medium intensity transition, below which intensity is 'Medium' and above is 'High'.

There tunable parameters for time durations are:

- **POWER_ENVELOPE_TIME**: Time envelope used to smoothen the power measurement.
- **POWER_ATTACK_TIME**: Power measurement attack time.

To tune other tunable parameters such as wind thresholds, use a wind generator and apply different wind speeds.

By viewing the wind intensity plot, choose the thresholds where the transitions occur.

When intensity categorization is not enabled in the build, release thresholds are not applicable, and intensity transitions will solely be based on **MED_WIND_THRESHOLD** and **HIGH_WIND_THRESHOLD**

NOTE Do not change the **POWER_ENVELOPE_TIME** parameter. If there is a requirement to track large increases in volume faster, optionally modify the **POWER_ATTACK_TIME** parameter. However, if this parameter is not changed carefully, it might break the measurement.

Figure 11-20 shows the graphical view.

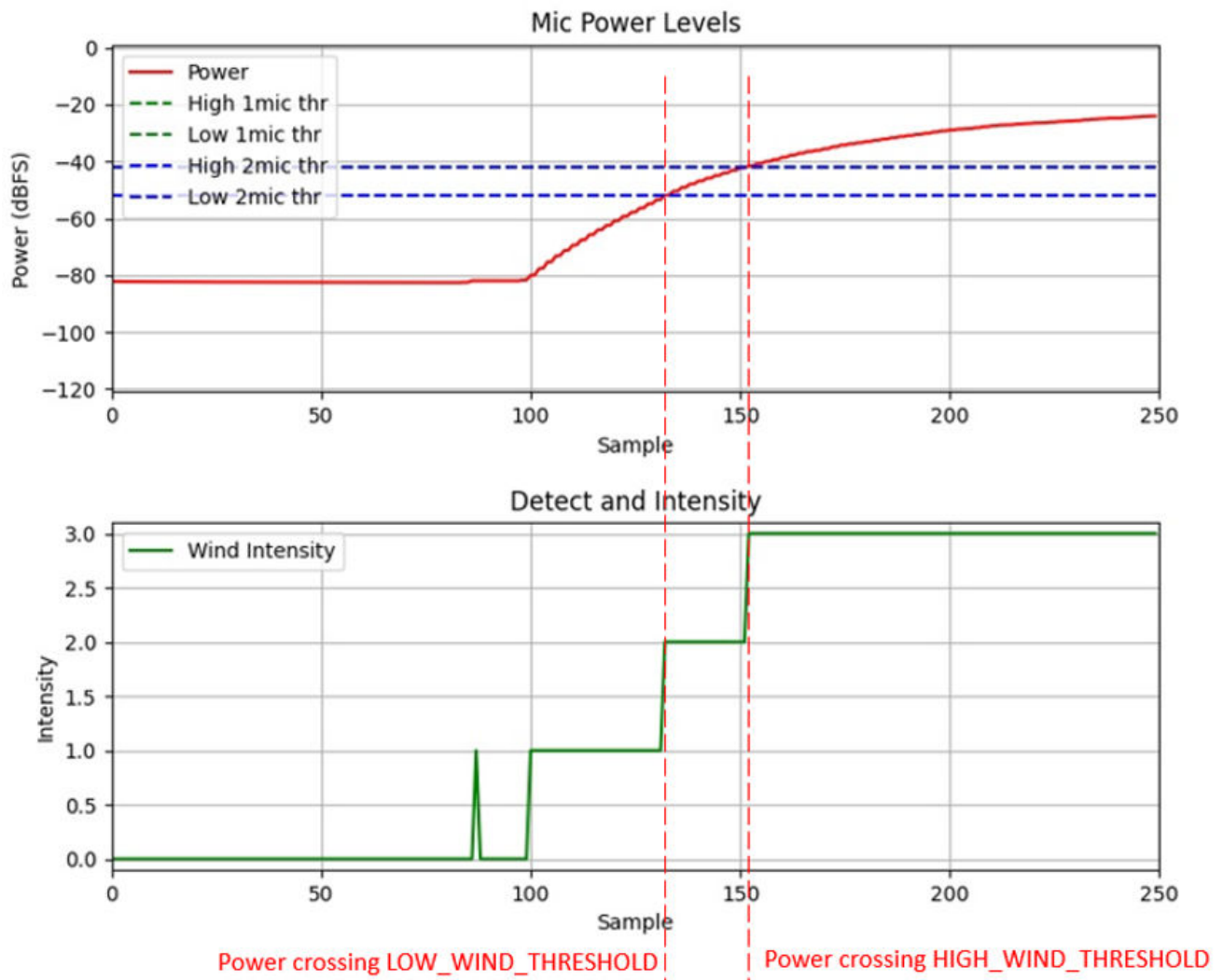


Figure 11-20 Wind intensity levels

11.2.4.3 Wind mitigation parameters tuning

When intensity categorization is enabled in the build, application would trigger wind noise mitigation by reading relevant parameters from WND capability and overriding them to AHM capability before updating it's sysmode. There are twelve tunable parameters for detecting wind intensity. These parameters are common for both 1-mic and 2-mic modes.

Table 11-6 lists the parameters used for tuning the wind mitigation parameters.

Table 11-6 Wind mitigation parameters

Mode	Parameter	Unit	Range	Description
1-Mic/2-Mic	LOW_WIND_FF_RAMP_DURATION	sec	0 to 10	Duration of the FF ramp into Low windy mode.
1-Mic/2-Mic	LOW_WIND_FB_RAMP_DURATION	sec	0 to 10	Duration of the FB ramp into Low windy mode. See Gain payload .

Table 11-6 Wind mitigation parameters (cont.)

Mode	Parameter	Unit	Range	Description
1-Mic/2-Mic	LOW_WIND_FF_FINE_GAIN	-	-	FF fine gain in Low intensity wind. See Gain payload .
1-Mic/2-Mic	LOW_WIND_FB_FINE_GAIN	-	-	FB fine gain value in Low intensity wind. See Gain payload .
1-Mic/2-Mic	MEDIUM_WIND_FF_RAMP_DURATION	sec	0 to 10	Duration of the FF ramp into Medium windy mode.
1-Mic/2-Mic	MEDIUM_WIND_FB_RAMP_DURATION	sec	0 to 10	Duration of the FB ramp into Medium windy mode.
1-Mic/2-Mic	MEDIUM_WIND_FF_FINE_GAIN	-	-	FF fine gain value in medium intensity wind. See Gain payload .
1-Mic/2-Mic	MEDIUM_WIND_FB_FINE_GAIN	-	-	FB fine gain value in medium intensity wind. See Gain payload .
1-Mic/2-Mic	HIGH_WIND_FF_RAMP_DURATION	sec	0 to 10	Duration of the FF ramp into High windy mode.
1-Mic/2-Mic	HIGH_WIND_FB_RAMP_DURATION	sec	0 to 10	Duration of the FB ramp into High windy mode.
1-Mic/2-Mic	HIGH_WIND_FF_FINE_GAIN	-	-	FF fine gain value in High intensity wind. See Gain payload .
1-Mic/2-Mic	HIGH_WIND_FB_FINE_GAIN	-	-	FB fine gain value in High intensity wind. See Gain payload .

Gain payload

Gain can be one of:

- Absolute value
- Shift from static gain in steps of 6 dB.

Table 11-7 Gain payload

Bits range	Description
31:24	Gain type 0x00 = Absolute value 0x80 = Shift from static gain
15:8	Shift value from static gain
7:0	Absolute value

11.2.4.4 Configure wind messaging

The Wind Noise Detect (WND) capability sends unsolicited messages to the application for attacking and releasing wind detection. These messages are controlled by a state machine internal to the WND capability. To control the state machine behavior, tune the `xMIC_MESSAGE_ATTACK` and `xMIC_MESSAGE_RELEASE` parameters. The parameters are similar for both 1-mic and 2-mic modes.

Table 11-8 lists the parameters for configuring wind messaging.

Table 11-8 Wind messaging parameters

Modes	Parameters	Unit	Range	Description
1-Mic	1MIC_MESSAGE_ATTACK	sec	0 to 20	Duration of 1-mic wind detection required to send a detection notification
1-Mic	1MIC_MESSAGE_RELEASE	sec	0 to 20	Duration of the absence of 1-mic wind detection required to send a detection cleared notification
2-Mic	2MIC_MESSAGE_ATTACK	sec	0 to 20	Duration of 2-mic wind detection required to send a detection notification
2-Mic	2MIC_MESSAGE_RELEASE	sec	0 to 20	Duration of the absence of 2-mic wind detection required to send a detection cleared notification

The `xMIC_MESSAGE_ATTACK` parameter configures the duration of wind detection required to send a wind detection notification to the application. Similarly, the `xMIC_MESSAGE_RELEASE` parameter configures the duration of absence wind required to send a detection cleared notification.

Figure 11-21 shows the graphical view.

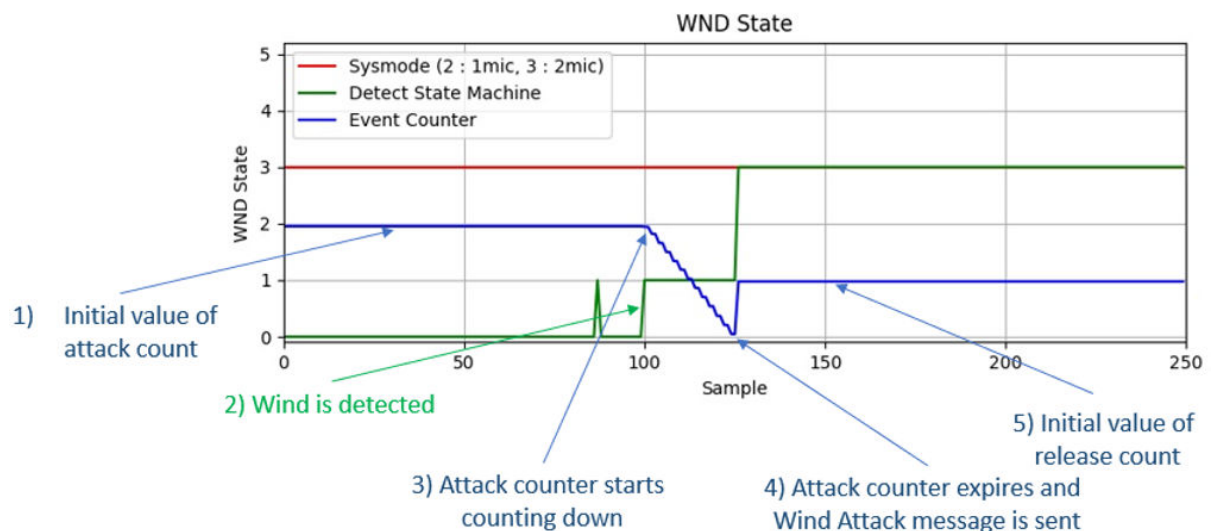


Figure 11-21 Wind state

11.3 Auto Transparency feature

The Auto Transparency (ATR) feature supports detection of user voice or self-speech and switches the ANC mode to *configured mode* during the speech. The ATR feature automatically switches to transparent mode (attack) when the user speaks and switches back to the previous mode (release) when no user speech is detected. The ATR feature deploys Bone conducting microphone to detect self speech.

- NOTE**
- The ATR audio graph runs only on the primary earbud and detects self-speech. Based on the trigger or release, the application changes the ANC mode in a synchronized way between the buds.
 - ATR is applicable to Standalone and Music concurrency use cases.

11.3.1 ATR_VAD capability

The Auto Transparency Voice Activity Detection (ATR_VAD) capability provides 1-mic algorithm for detecting the presence of voice activity. When voice is detected, an unsolicited message is sent to the application allowing for a synchronized mode change between the earbuds. The algorithm returns a likelihood of voice activity, which is held at a threshold to get instantaneous. detection triggers. Instantaneous triggers along with hold and release timers return voice activity detections.

The timers associated with each detection such that voice activity (or the absence of voice activity) have to be detected for a particular duration before any message is sent to the application.

11.3.2 Build and configure the ATR_VAD capability

Before you begin, ensure that the ANC and Adaptive ANC features are enabled, as described in [Enable ANC features in the Earbuds application](#). To build and configure the ATR_VAD capability, add the required definitions to the project in Qualcomm MDE.

Enable the ATR_VAD capability in the Earbuds project

ATR works with Adaptive ANC Feature license (subs7_config3.htf).

To enable the ATR_VAD capability:

1. Go to **MDE Projects > General > DEFS**, and then add the `ENABLE_AUTO_AMBIENT` and `ENABLE_SELF_SPEECH` definitions in the earbud project properties.
2. Check that the `download_atr_vad.edkcs` file is present in the `ro_fs` folder. If it is not present, add it manually from the `...\audio\<chip variant>\kalimba_ROM_xxxx\kymera\prebuilt_dkcs\maor_rom_release\...` folder.
3. The `<ADK path>\src\domains\audio\kymera\kymera_config.h` file has the following default configurations for the ATR_VAD feature:

```
#define appConfigAutoAmbientEnable()          (TRUE)
#define appConfigAutoAmbientMode()
(anc_mode_4)
#define appConfigSelfSpeechDetectMic()
(appConfigMicInternalBCM())
```

Debug with Pydbg

[Table 11-9](#) lists the Pydbg commands for testing the ATR_VAD capability.

Table 11-9 Pydbg ATR_VAD test commands

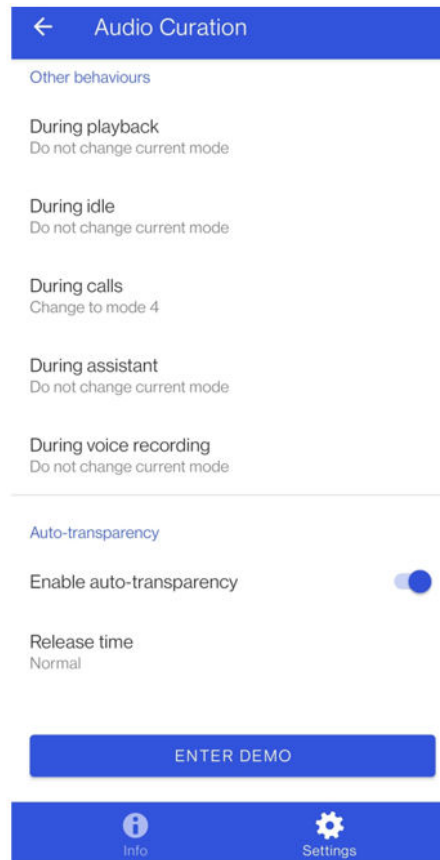
Test interfaces	Usage
<code>EarbudTest_EnableAutoTransparency()</code>	API to enable or start the Auto Transparency feature
<code>EarbudTest_DisableAutoTransparency()</code>	API to disable the Auto Transparency feature

Table 11-9 Pydbg ATR_VAD test commands (cont.)

Test interfaces	Usage
<code>EarbudTest_IsAutoTransparencyEnabled()</code>	API to check if Auto Transparency is enabled
<code>EarbudTest_GetAtrReleaseDuration()</code>	API to get the ATR Release duration
<code>EarbudTest_SetAtrReleaseDuration()</code>	API to set the ATR Release duration

Voice activity detection using GAIA Application

1. Start the GAIA Application for earbuds.
2. Select the earbuds to connect to, and then go to **Audio Curation > Demo Mode**.
3. Using the User configuration options, complete the following tasks:
 - Enable/Disable Auto transparency.
 - Configure ATR Release time.
4. The mode changes on self speech detect and release are visible in the **Demo** mode.

**Figure 11-22 Enabling Auto Transparency feature**

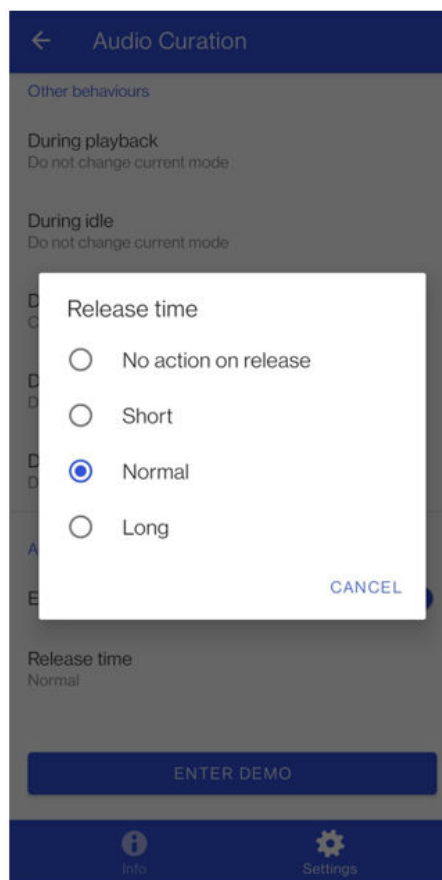


Figure 11-23 ATR Release time options

11.3.3 Tune the Auto Transparency feature

The Auto-Transparency feature comprises of two capabilities, *PEQ* and *ATR_VAD* in the audio chain.

11.3.3.1 VAD algorithm tuning

The sensitivity of the VAD algorithm to the presence of non-stationary noise (speech) can be tuned. This is divided into three stages: tuning the power envelope, tuning the detection sensitivity, and tuning for rejection.

Tuning the power envelope

The power envelope includes three signals, shown in Figure 11-24.

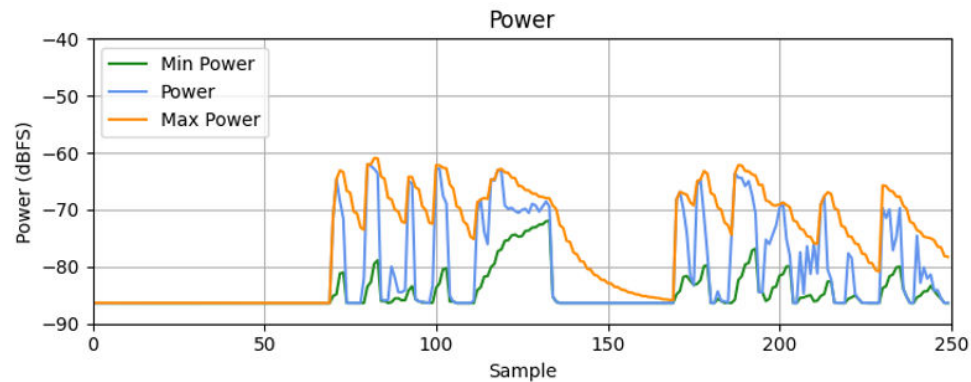


Figure 11-24 VAD power envelopes

- Power: Smoothened instantaneous power.
- Min power: Minimum envelop of the power signal. It closely tracks the stationary noise power envelop.
- Max power: Maximum envelop of the power signal. It closely tracks the non-stationary noise power envelop.

The power signal is smoothed using `VAD_ENVELOPE_TIME`. When this value is increased, the overall algorithm is less reactive to change in signal power and vice versa.

The max power is *decay* filtered using `VAD_RELEASE_TIME`, which if increased reduces sensitivity to non-stationary (loud) noise. Min power is *attack* filtered using `VAD_ATTACK_TIME`, which if increased reduces sensitivity to the stationary or environmental noise.

Tuning the VAD detection sensitivity

To tune detections, use the `VAD_RATIO` parameter. The larger the ratio, the less sensitive the algorithm is to differences between stationary and non-stationary signal levels.

Tuning VAD rejections

After VAD is detected in previous stage, the algorithm can still reject the detection:

- If the max power is less than `VAD_PWR_THRESHOLD`, the detection is rejected.
- If the envelope size is less than `VAD_ENV_THRESHOLD`, the detection is rejected. This value can be tuned (similar to AANC) to accommodate a certain amount of non-stationary noise before detection by increasing the threshold.

Table 11-10 lists the VAD tuning parameters.

Table 11-10 Tunable parameters

Parameter	Unit	Range	Description
VAD_ATTACK_TIME	sec	0.01 to 5	Attack time for VAD minimum power envelope
VAD_RELEASE_TIME	sec	0.01 to 5	Release time for VAD maximum power envelope
VAD_ENVELOPE_TIME	sec	0.001 to 5	Envelope smoothing time for VAD power envelope
VAD_RATIO	-	0 to 1	VAD detection ratio
VAD_PWR_THRESHOLD	dBFS	-100 to 0	VAD detection threshold
VAD_ENV_THRESHOLD	dBFS	-50 to 50	VAD power envelope threshold

11.3.3.2 VAD detection flag tuning

The VAD algorithm outputs a likelihood which is smoothed and a threshold to get an instantaneous trigger. The trigger is then passed through hold logic to give a final detection flag.

Table 11-11 shows the plots for likelihood (smoothed), triggers, and detection.

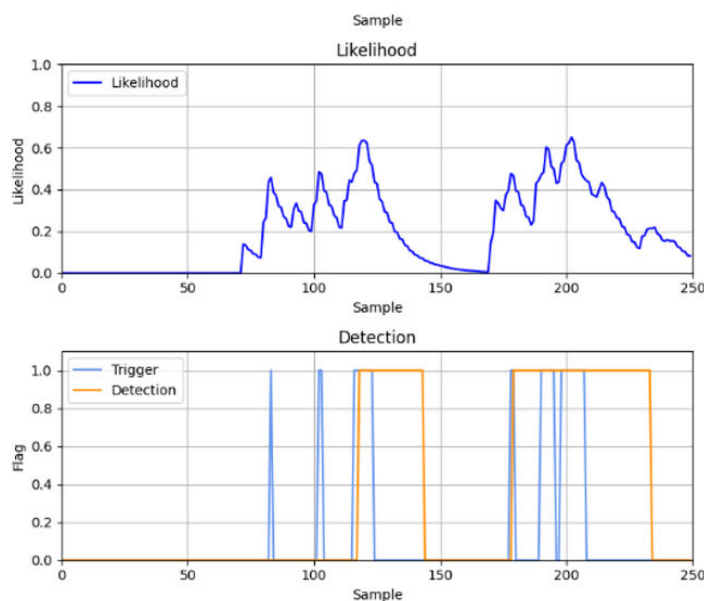


Figure 11-25 VAD likelihood, triggers, and detections

The instantaneous likelihood value is smoothed using `SMOOTH_ATTACK_TIME` and `SMOOTH_RELEASE_TIME`. The smoothed likelihood value is threshold using `DET_THRESHOLD` and to get instantaneous trigger.

The instantaneous trigger is held to get detection triggers. If the instantaneous trigger is held high continuously for `HOLD_ATTACK_TIME`, VAD detection flag is set. If instantaneous trigger is held low continuously for `HOLD_RELEASE_TIME`, VAD detection flag is cleared.

Table 11-11 lists the tunable parameters for VAD detection flag tuning.

Table 11-11 VAD detection flag tuning parameters

Parameters	Unit	Range	Description
SMOOTH_ATTACK_TIME	sec	0 to 60	Detection value smoothing attack time
SMOOTH_RELEASE_TIME	sec	0 to 60	Detection value smoothing release time
HOLD_ATTACK_TIME	sec	0 to 60	Detection value hold attack time
HOLD_RELEASE_TIME	sec	0 to 60	Detection value hold release time
DET_THRESHOLD	sec	0 to 1	Detection threshold

11.3.3.3 ATR_VAD messaging tuning

The ATR_VAD capability sends unsolicited messages to the application for attacking and releasing VAD detection. These messages are controlled by a state machine internal to the ATR_VAD capability.

To control the state machine behavior, tune the MSG_ATTACK_TIME and MSG_XXXXX_RELEASE_TIME parameters, shown in Figure 11-26.

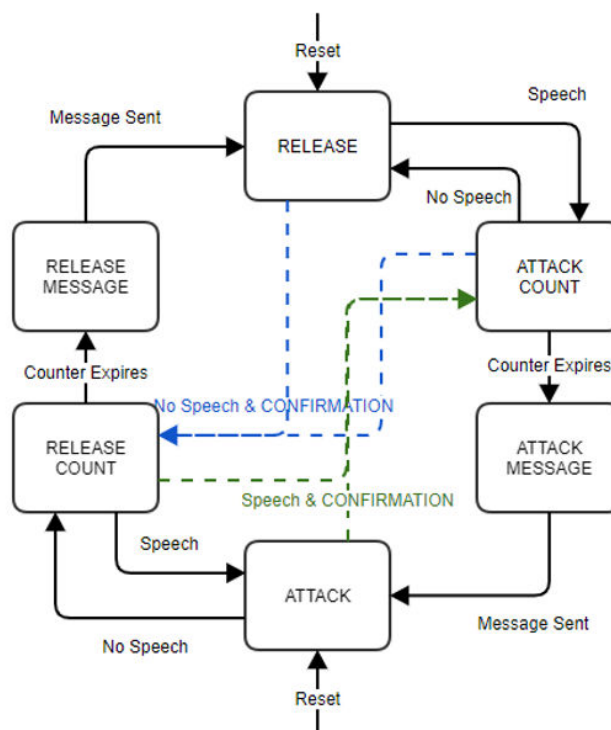


Figure 11-26 ATR_VAD messaging state machine

The MSG_ATTACK_TIME parameter configures the duration of voice detection required to send a VAD detection notification to the application. Similarly, the MSG_XXXXX_RELEASE_TIME parameter configures the duration of absence of voice activity required to send a detection cleared notification.

Table 11-12 lists the parameters for ATR_VAD messaging tuning.

Table 11-12 ATR_VAD messaging tunable parameters

Parameters	Unit	Range	Description
MSG_ATTACK_TIME	sec	0-60	Duration of voice activity detection required to send a detection notification
MSG_SHORT_RELEASE_TIME	sec	0-60	Duration of the absence of voice activity detection required to send a detection cleared notification in short release time
MSG_NORMAL_RELEASE_TIME	sec	0-60	Duration of the absence of voice activity detection required to send a detection cleared notification in normal release time
MSG_LONG_RELEASE_TIME	sec	0-60	Duration of the absence of voice activity detection required to send a detection cleared notification in long release time

11.3.3.4 ATR_VAD false triggers

The ATR VAD capability is susceptible to a certain false triggers. Some of these triggers can be tuned against at the cost of reduced sensitivity to self-speech. The performance of this is dependent on other system factors, for example, the ambient environment rejection characteristic of the BCM.

The false triggers are:

- Extremely loud ambient voice
- Humming of songs
- A2DP playback at extremely loud and uncomfortable levels
- Use of an electric toothbrush
- Long or speech-like voiced activity, for example certain coughs

11.4 Adverse Acoustic Event Handler (AAH) feature

The Adverse Acoustic Event Handler (AAH) capability detects the presence of adverse acoustic events during user actions such as rubbing the ear, chewing, or coughing, when using the earbuds. When an event is detected, the feed-forward gain and/or the feedback gain are reduced to avoid clipping the output of the ANC hardware.

11.4.1 Adverse Acoustic Handling capability system

The AAH capability system includes the Adverse Acoustic Handler (AAH), ANC Hardware Manager (AHM), the DSP, and the ANC hardware.

The principal function of the AAH capability and its connections within the system are shown in [Figure 11-27](#).

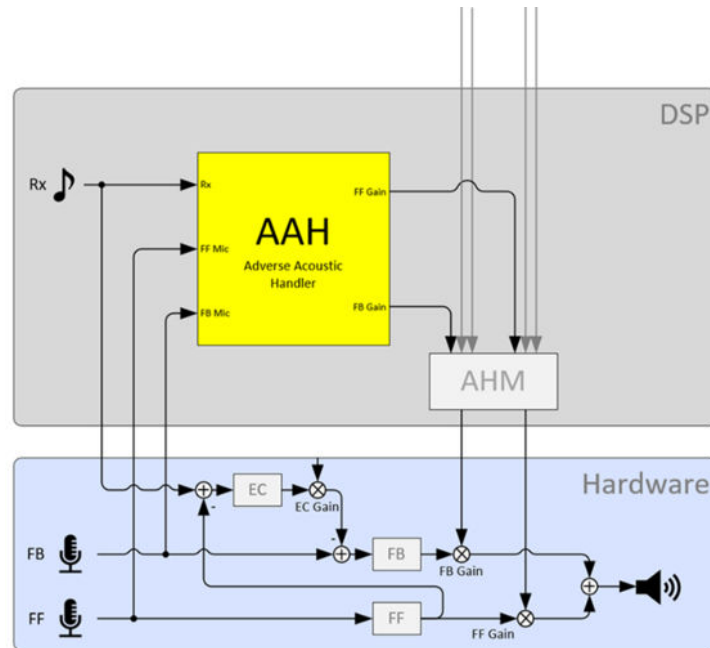


Figure 11-27 AAH capability system block diagram

The AAH detects clipping on internal ANC Hardware nodes by approximating the ANC filters. On clipping events, the AAH sends gain reductions to the AHM which then updates the post filter gains of the ANC hardware.

The ANC hardware is modelled in the DSP. With simplified filters (proxy filters), the internal gain structure is simulated. The DSP is fed with the same signals the ANC Hardware is connected to (Rx, FF Mic, FB Mic). The DSP routing is explained in [Tune the Adverse Acoustic Handling capability](#).

With the proxy filters for feed-forward, feedback and Echo Canceller, the AAH clipping estimation knows, when clipping is likely to occur in the ANC hardware. When clipping is detected, the respective gains are reduced according to the predicted amount of overload.

The capability ramps the gains, following an Attack - Hold - Release characteristic. The gain reduction is released again after a programmable hold time.

11.4.2 Build and configure the Adverse Acoustic Handling capability

Before you begin, ensure that the ANC and Adaptive ANC features are enabled, as described in [Enable ANC features in the Earbuds application](#). To build and configure the AAH capability, add the required definitions to the project in Qualcomm MDE.

Enable the AAH capability in the Earbuds project

Go to **MDE Projects > General > DEFS**, and then add the `ENABLE_ANC_AA_H` and `ENABLE_ANC_AA_H_CC` definitions in the earbud project properties.

11.4.3 Tune the Adverse Acoustic Handling capability

Figure 11-28 shows the parameters that can be tuned and where they are effective in the adverse acoustic handling.

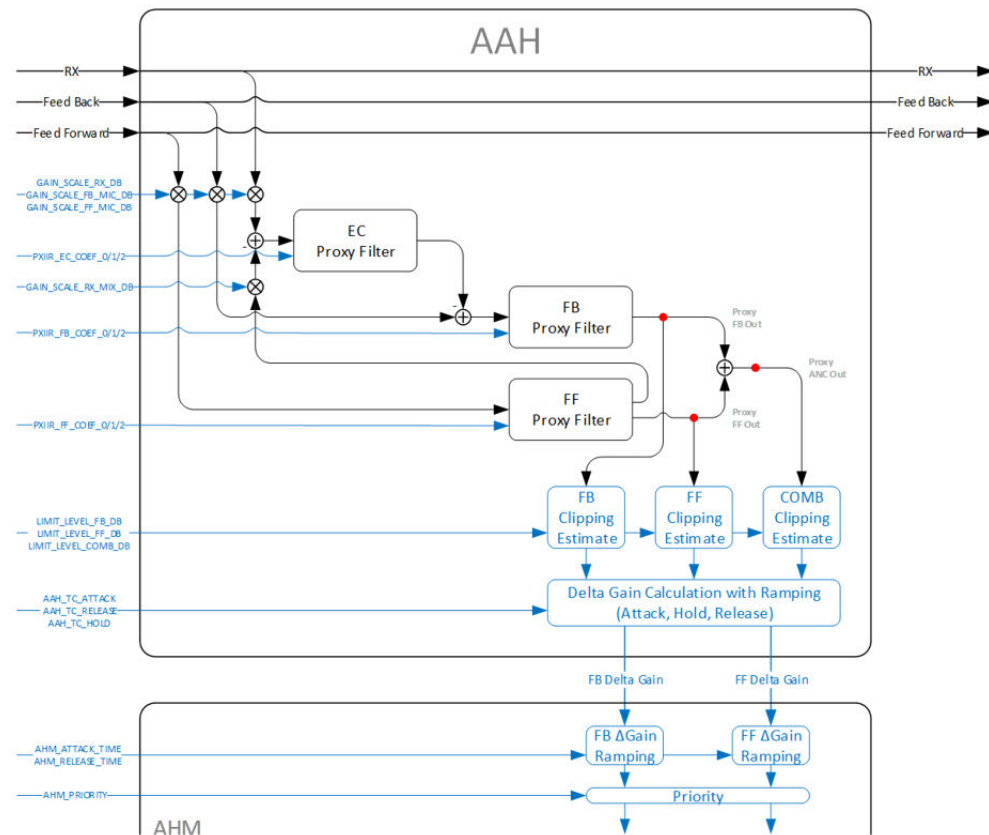


Figure 11-28 AAH algorithm block diagram with parameters

11.4.3.1 AAH tuning parameters in AHM

The tunable parameters for the AHM capability are listed in Table 11-13.

Table 11-13 AHM tuning parameters

Name	Offset	Format	Default value	Description
AHM_PRIORITY	0	QM.0	5	AHM Gain Priority of the AAH client
AHM_ATTACK_TIME	1	Q1.N	0.125	Attack coefficient for AHM shared gain ramping
AHM_RELEASE_TIME	2	Q1.N	0.025	Release coefficient for AHM shared gain ramping

11.4.3.2 AAH algorithm tuning parameters

Table 11-14 lists the parameters for tuning the AAH algorithm.

Table 11-14 AAH algorithm tuning parameters

Name	Offset	Format	Default value	Description
AAH_CONFIG	3	QM.0	0	Configuration flags: 0x0001- lib enable: Configures the capability so that the library code is enabled
GAIN_SCALE_RX_DB	4	QM.0	0	RX gain scaling in 60ths of a dB
GAIN_SCALE_FB_MIC_DB	5	QM.0	0	Feedback mic gain scaling in 60ths of a dB
GAIN_SCALE_FF_MIC_DB	6	QM.0	0	Feedforward mic gain scaling in 60ths of a dB
GAIN_SCALE_RX_MIX_DB	7	QM.0	0	RX-mix gain scaling in 60ths of a dB
LIMIT_LEVEL_FB_DB	8	QM.0	-180	Feedback Limit threshold level in 60ths of a dB
LIMIT_LEVEL_FF_DB	9	QM.0	-180	Feed-forward Limit threshold level in 60ths of a dB
LIMIT_LEVEL_COMB_DB	10	QM.0	-180	Combined (Feedback and Feed-forward) Limit threshold level in 60ths of a dB

11.4.3.3 AAH timing tuning parameters

Table 11-15 lists the parameters for tuning the AAH timing.

Table 11-15 AAH timing tuning parameters

Name	Offset	Format	Default value	Description
AAH_TC_ATTACK	11	QM.0	0	AAH attack time constant in μ sec
AAH_TC_RELEASE	12	QM.0	10000	AAH release time constant in μ sec
AAH_TC_HOLD	13	QM.0	10000	AAH hold time constant in μ sec

11.4.3.4 AAH proxy filter tuning parameters

The proxy filter coefficients must be created together with the ANC Hardware filter coefficients in the ANC Designer Tool. When saving the .htf file for a filter, the coefficients for the proxy filter are included in the same .htf file, but must be extracted for use in QACT, or as part of the include files for the build.

Example of proxy filter coefficients in the *.htf file:

```
# Adverse acoustic handler filter
# Workspace: C:\xxx\xxx.pws
```

```
# Snapshot: Filter xx
0x206800 = [ 01 10 0E 00 09 00 E0 7F 00 09 80 08 D0 7F 10 03 E0 7F E0 7F 70
2D F0 32 C0 7F 80 7D 80 7F C0 7F F0 08 A0 D8 00 73 D0 CF A0 7F ]
```

Table 11-16 lists the parameters used for proxy filter tuning.

Table 11-16 ANC proxy filter tunable parameters

Name	Offset	Format	Default value	Description
PXIIR_EC_COEF_0	14	QM.0	0x7fff0800	EC approx IIR coefficient 0
PXIIR_EC_COEF_1	15	QM.0	0x00000000	EC approx IIR coefficient 1
PXIIR_EC_COEF_2	16	QM.0	0x00000000	EC approx IIR coefficient 2
PXIIR_FB_COEF_0	17	QM.0	0x7fff0800	FB approx IIR coefficient 0
PXIIR_FB_COEF_1	18	QM.0	0x00000000	FB approx IIR coefficient 1
PXIIR_FB_COEF_2	19	QM.0	0x00000000	FB approx IIR coefficient 2
PXIIR_FF_COEF_0	20	QM.0	0x7fff0800	FF approx IIR coefficient 0
PXIIR_FF_COEF_1	21	QM.0	0x00000000	FF approx IIR coefficient 1
PXIIR_FF_COEF_2	22	QM.0	0x00000000	FF approx IIR coefficient 2

11.4.4 Statistics of Adverse Acoustic Event Handler capability

Table 11-17 lists the AAH statistics, to be read with QACT.

Table 11-17 AAH statistics

Name	Offset	Format	Description
CUR_MODE	0	QM.0	System Mode
EC_DELTA_GAIN	1	QM.0	Echo canceller path shared delta gain (reserved, always 1.0)
FB_DELTA_GAIN	2	QM.0	Feedback path shared delta gain
FF_DELTA_GAIN	3	QM.0	Feed-forward path shared delta gain

11.5 DSP SW based leak-through feature

The DSP SW based leak-through functionality adds the ability to leak-through audio from the microphones to the speaker. The functionality is enabled by using the existing AEC-Ref capability's sidetone path where the microphone gets connected to the speaker or DAC through the AEC Ref capability.

NOTE In case of DSP SW based leak-through, the latency between the microphone and the speaker is in the order of few milliseconds. Qualcomm recommend using software based leak-through only if ANC is not enabled in the build configuration of the product.

11.5.1 Pydbg commands to control DSP-based audio leak-through

By default, the Audio leak-through using AEC Ref capability is disabled. Different button events are mapped for different functionalities of DSP based leak-through. For supported leak-through events, refer to the `x_button.buttonxml` file. The Pydbg commands listed in this section can also be used.

Pydbg commands: Earbuds Application

The following Pydbg commands are provided for leak-through operations on the Earbuds Application:

- `apps1.fw.call.EarbudTest_EnableLeakthrough()`
- `apps1.fw.call.EarbudTest_DisableLeakthrough()`
- `apps1.fw.call.EarbudTest_ToggleLeakthroughOnOff()`
- `apps1.fw.call.EarbudTest_SetLeakthroughMode(leakthrough_mode)`
- `apps1.fw.call.EarbudTest_SetNextLeakthroughMode()`
- `apps1.fw.call.EarbudTest_GetLeakthroughMode()`
- `apps1.fw.call.EarbudTest_IsLeakthroughEnabled()`

Pydbg commands: Headset Application

The following Pydbg commands are provided for leak-through operations on the Headset Application:

- `apps1.fw.call.HeadsetTest_SetLeakthroughEnable()`
- `apps1.fw.call.HeadsetTest_SetLeakthroughDisable()`
- `apps1.fw.call.HeadsetTest_SetLeakthroughToggleOnOff()`
- `apps1.fw.call.HeadsetTest_SetLeakthroughMode(leakthrough_mode)`
- `apps1.fw.call.HeadsetTest_SetLeakthroughNextMode()`
- `apps1.fw.call.HeadsetTest_IsLeakthroughEnabled()`

11.5.2 Enable, disable, or toggle audio leak-through functionality

11.5.2.1 Enable audio leak-through in standby mode

By default, the sidetone path/leak-through is not enabled. To enable or disable audio leak-through, use the button event or Pydbg commands when the earbuds are in standby mode.

To enable and verify sidetone:

1. Open the Earbud or Headset workspace from installed ADK path, for example
`QCC514x_QCC304x.SRC.1.0\earbud\workspace\QCC3040-AA_DEV-BRD-R2-AA\earbud.x2w` or `QCC514x_QCC304x.SRC.1.0\QCC514x_QCC304x.SRC.1.0\headset\workspace\QCC3040-AA_DEV-BRD-R2-AA\headset.x2w`
2. Go to **Project Settings > DEFS**, and then add the `INCLUDE_KYMER_AEC` and `ENABLE_AEC_LEAKTHROUGH` macros.
3. Build and Deploy the generated image on the each of the earbuds.

4. Connect the noise audio source to the microphone (MIC1 in CF376 development board).
5. Take the earbuds out of the case, or power on the headset.
6. Trigger leak-through by using the `apps1.fw.call Pydbg` command.
7. Establish the AEC Ref capability chain by using either the `EarbudTest_ToggleLeakthroughOnOff` command, or `HeadsetTest_SetLeakthroughToggleOnOff` command.
8. In the QACT tool, validate the standalone AEC Ref capability chain created.
9. Start streaming leak-through noise from the audio source. The injected noise should be audible at the earbud or headset.
10. Using the Pydbg command-line, run the `apps1.fw.call` command. Use either the `EarbudTest_ToggleLeakthroughOnOff` command, or `HeadsetTest_SetLeakthroughToggleOnOff` command for destroying the AEC Ref capability chain.

11.5.2.2 Toggle audio leak-through with an active A2DP connection

To toggle audio leak-through during an active A2DP connection, use either the button event or the appropriate Pydbg commands.

To toggle audio leak-through with an active A2DP connection:

1. Enable and verify sidetone, as described in [Enable audio leak-through in standby mode](#).
2. Connect the earbud or headset to the QACT tool. Verify that the A2DP leak-through chain is created.
3. Inject noise from noise source to earbud or headset. The injected noise should be audible at the earbud or headset along with A2DP audio.

11.5.2.3 Toggle audio leak-through with an active SCO connection

To toggle audio leak-through during an active SCO connection, use either the button event or the appropriate Pydbg commands.

To toggle audio leak-through during an active SCO connection:

1. Enable and verify sidetone, as described in [Enable audio leak-through in standby mode](#).
2. Connect earbud/headset to QACT tool and verify the SCO chain is created.
3. Inject noise from noise source to earbuds/headset. The injected noise should be audible at the earbud/headset along with SCO audio.

11.5.2.4 Transition between audio sources

Transition from an audio source to another audio source is supported. For example, the transition from standalone leak-through to A2DP leak-through/SCO leakthrough and vice versa is possible and the leak-through state is retained. However, transition from standalone leak-through to Voice UI leak-through and vice versa is not supported.

To transition from standalone leak-through to A2DP leak-through:

1. Enable and verify sidetone, as described in [Enable audio leak-through in standby mode](#).
2. Stream A2DP audio using connected audio gateway(AG) for example, mobile handset and verify the A2DP leakthrough chain is created in QACT tool. The injected noise should be audible at the left/right earbud along with the A2DP audio.

11.5.2.5 Set audio leak-through mode (UCID)

Three leak-through modes, `LEAKTHROUGH_MODE_1`, `LEAKTHROUGH_MODE_2`, and `LEAKTHROUGH_MODE_3`, are supported to configure different microphone gain, sidetone path gains and equalizer settings. When a Pydbg command is used for setting a leak-through mode, the relevant UCIDs are updated to the `AEC_REF` operator based on active cases such as standalone leak-through, A2DP leak-through, or SCO leak-through.

These UCIDs are defined in the `aec_reference_config.htf` file under `user_ps_filesystem` and located in the installed ADK path: `<ADK_Path>\QCC514x_QCC304x.SRC.1.0\headset\workspace\QCC5141-AA_DEV-BRD-R2-AA\filesystems\aec_reference_config.htf`.

By default, all UCIDs have the following default configuration:

- Sidetone path bypassed (disabled)
- Microphone gain at 0 dB
- PEQ filter disabled
- Sidetone path at 0 dB gain

During production, tune the configurations in UCIDs based on the requirements and form factor of the device. To tune the UCID configurations, use the QACT audio calibration tool.

To set the leak-through mode:

1. Enable and verify sidetone, as described in [Enable audio leak-through in standby mode](#).
2. In one of the earbud/headset, update the leak-through mode `LEAKTHROUGH_MODE_1` using the `apps1.fw.call Pydbg` command.
3. Configure the microphone gain, sidetone path gain, and equalizer settings by using either the `EarbudTest_SetLeakthroughMode(0)` command or the `HeadsetTest_SetLeakthroughMode(0)` command.
4. Start streaming leak-through noise from the audio source. The injected noise should be audible at the earbud/headset with right gain configurations.

11.5.2.6 Leak-through state and mode (UCID) synchronization between earbuds

The leak-through feature supports both state and mode synchronization. To set the state and mode, use Pydbg commands at the left/right earbud. When the state and mode are set, the local device and peer device are updated with reasonable latency.

Synchronize the leak-through state (enable or disable)

To synchronize leak-through state (enable or disable) on earbuds:

1. Open the earbud workspace from installed ADK path, for example, `QCC514x_QCC304x.SRC.1.0\earbud\workspace\QCC3040-AA_DEV-BRD-R2-AA\earbud.x2w`.
2. Go to **Project settings > DEFS**, and then add the `INCLUDE_KYMER_AEC` and `ENABLE_AEC_LEAKTHROUGH` macros.
3. Build and deploy the generated image on the each of the earbuds.
4. Connect noise audio source to microphone (MIC1 in CF376 development board) on the each of the earbuds.
5. Check the two devices are paired with one another using the `apps1.fw.call.appTestIsPeerPaired()` Pydbg command.
6. Trigger out of case event in both the devices using the `apps1.fw.call.appTestPhyStateOutOfCaseEvent()` Pydbg command.
7. Establish the AEC Ref capability chain on both local and remote earbuds. In one of the earbuds, trigger leak-through using the `apps1.fw.call.EarbudTest_ToggleLeakthroughOnOff()` Pydbg command.
8. In the QACT tool, validate the standalone AEC Ref capability chain created on the each of the earbuds.
9. Start streaming leak-through noise from the audio source. The injected noise should be audible at the left and right earbuds.

Synchronize the leak-through mode

To synchronize the leak-through mode between earbuds:

1. Open the earbud workspace from installed ADK path, for example, `QCC514x_QCC304x.SRC.1.0\earbud\workspace\QCC3040-AA_DEV-BRD-R2-AA\earbud.x2w`.
2. Go to **Project settings > DEFS**, and then add the `AEC_LEAKTHROUGH_ENABLED` macro.
3. In `aec_reference_config.htf` file, configure microphone gain, sidetone path gain and equalizer settings.
4. Build and Deploy the generated image on the each of the earbuds.
5. Connect noise audio source to microphone (MIC1 in CF376 development board) on the each of the earbuds.
6. Check the two devices are paired with one another using the `apps1.fw.call.appTestIsPeerPaired()` Pydbg command.

7. Trigger out of case event in both the devices using the `apps1.fw.call.appTestPhyStateOutOfCaseEvent()` Pydbg command.
8. In one of the earbuds, trigger leak-through using the `apps1.fw.call` Pydbg command. Establish the AEC Ref capability chain on both local and remote earbuds by using either:
 - The `EarbudTest_ToggleLeakthroughOnOff()` command.
 - The button event by double tapping **Sys Control** button in CF376 development board.
9. In the QACT tool, validate the standalone AEC Ref capability chain created on the each of the earbuds
10. In one of the earbuds, update the leak-through mode `LEAKTHROUGH_MODE_1` by using the `apps1.fw.call` Pydbg command.
11. Configure the microphone gain, sidetone path gain, and equalizer settings by using the `EarbudTest_SetLeakthroughMode(0)` command.
12. Start streaming leak-through noise from the audio source. The injected noise should be audible at the left and right earbuds with the correct gain configurations.

11.6 NoiseID feature overview

The NoiseID feature enables users to classify the current environmental noise into two distinct categories by using one of two classifier algorithms: type-based classification or level-based classification. Customers are advised to use any one classifier algorithm at a time.

Type-based classification

This method classifies the environment into two noise categories, enabling the system to apply specific pre-designed filters according to the category. The categorization is based on features computed in the frequency domain by taking the ratio of the power of two different bins.

Level-based classification

This method classifies the environment into three noise intensity levels, enabling the system to apply specific pre-designed filters according to the intensity. The intensity level is computed using the power magnitude of frequency bins with certain weighting factors.

The image shows block diagram of the NoiseID state machine.

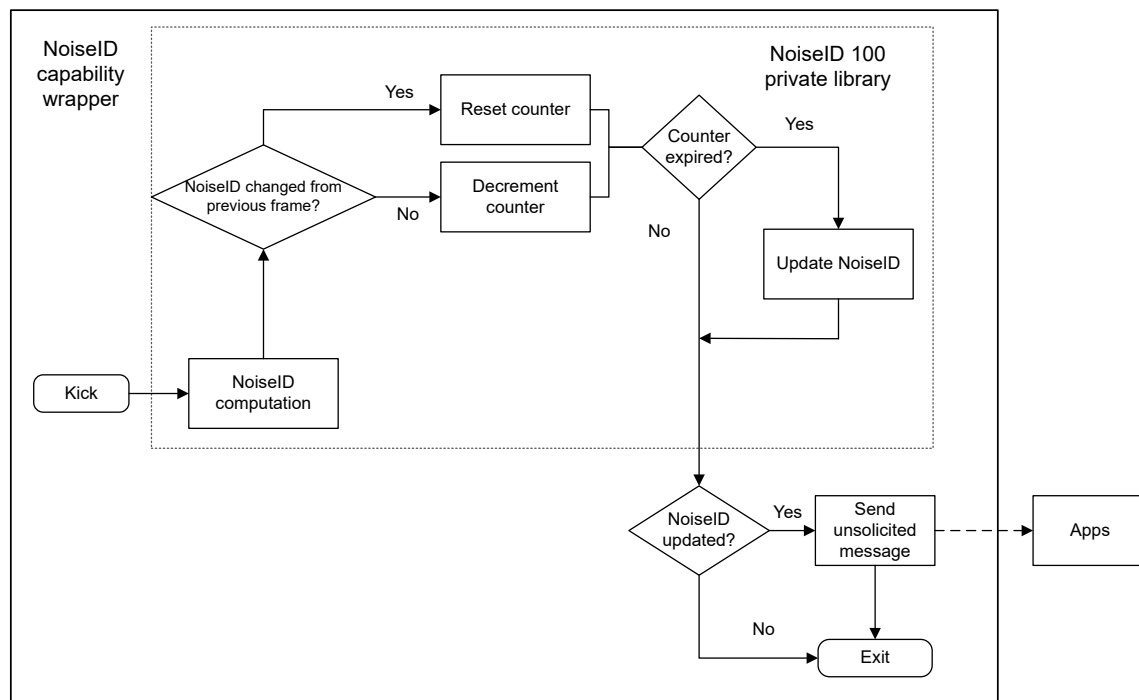


Figure 11-29 NoiseID state machine

The noise classification using Type-based classification is performed as follows:

- The power spectrum density of FF mic input is analyzed and divided into various frequency bins as shown in the figure.

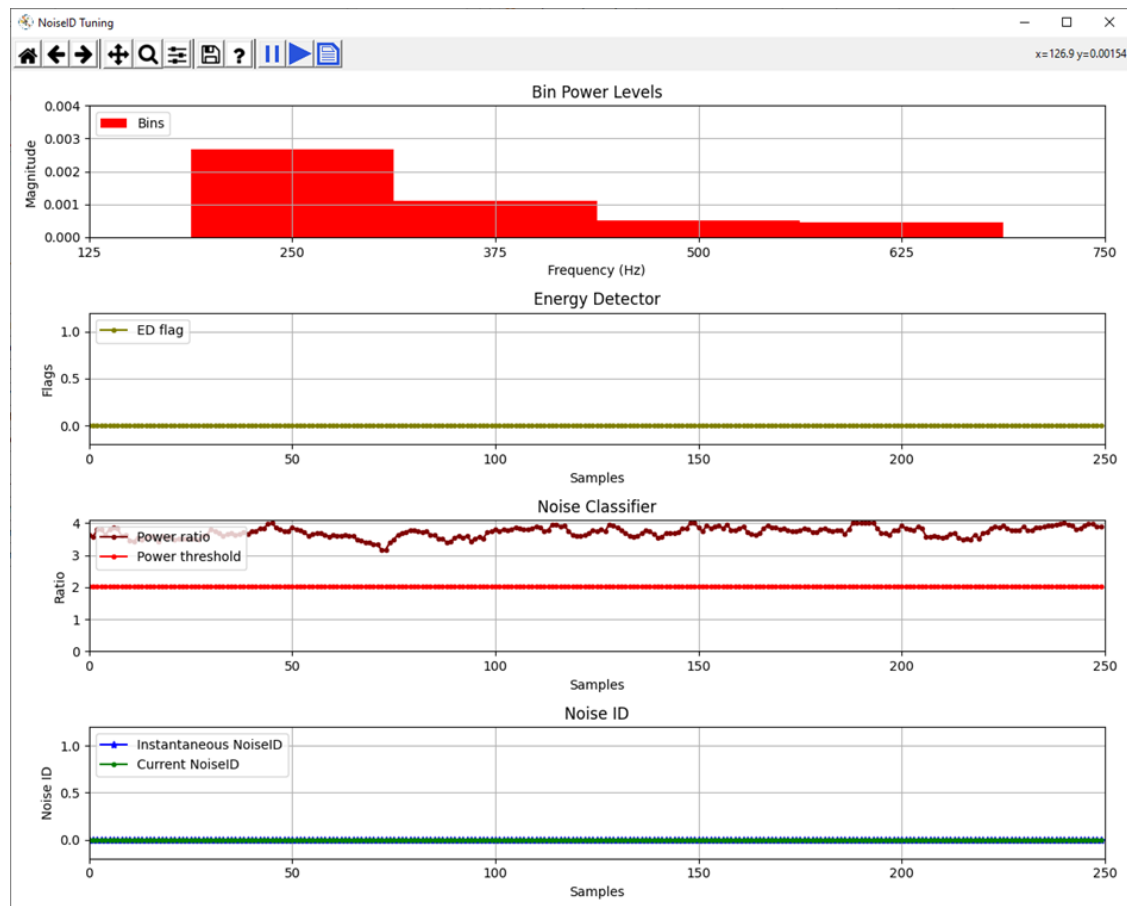


Figure 11-30 NoiselD capability for Type-based classification from the logger tool

- The average power of frequency bins 2 and 3 is compared with the frequency bins 4 and 5 and a power ratio is obtained.
 - Power ratio (P_r) = P_{b23}/P_{b45}
 - P_{b23} - average power of frequency bins 2 & 3
 - P_{b45} - average power of frequency bins 4 & 5
- The power ratio (P_r) is compared to a predefined power ratio threshold values (can be set using QACT)
- Once the power ratio (P_r) meets a predefined power ratio threshold for a set time or counter (defined using QACT), the noise category is switched to the other, which in-turn switches the ANC filter to the one tuned for this particular noise category.
- Noise category will not adapt if a non-stationary noise is detected in the environment. Detection of non-stationary noise is performed using Energy Detector(ED), similar to the one used in Adaptive ANC capability.

The noise classification using Level-based classification is performed as follows:

- The power spectral density of the FF microphone input is analyzed and segmented into various frequencies.
- The power in each bin is multiplied by its corresponding weighting filter response, then summed from bin 1 to bin 64 and smoothed using attack and decay time constants. Four predefined weighting filter responses exist, and one is chosen based on the weighting index specified in the tuning parameters.
- The resulting power is compared to predefined power threshold values (set using QACT).

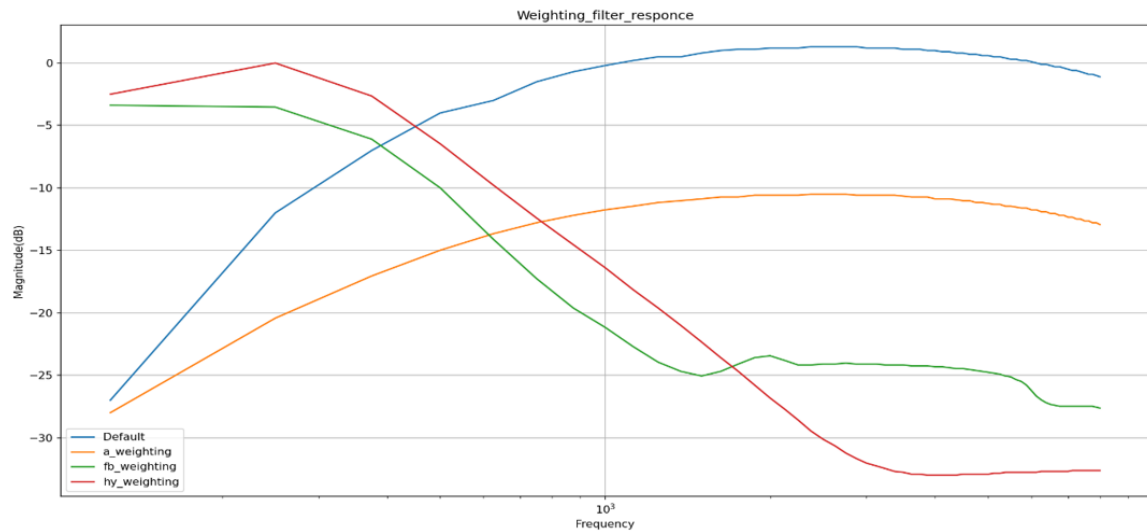


Figure 11-31 Weighting filter response curves

- When the power reaches a predefined threshold for a specified window size (set using QACT), the noise category switches to another, which in turn changes the ANC filter to the one tuned for this specific noise category.

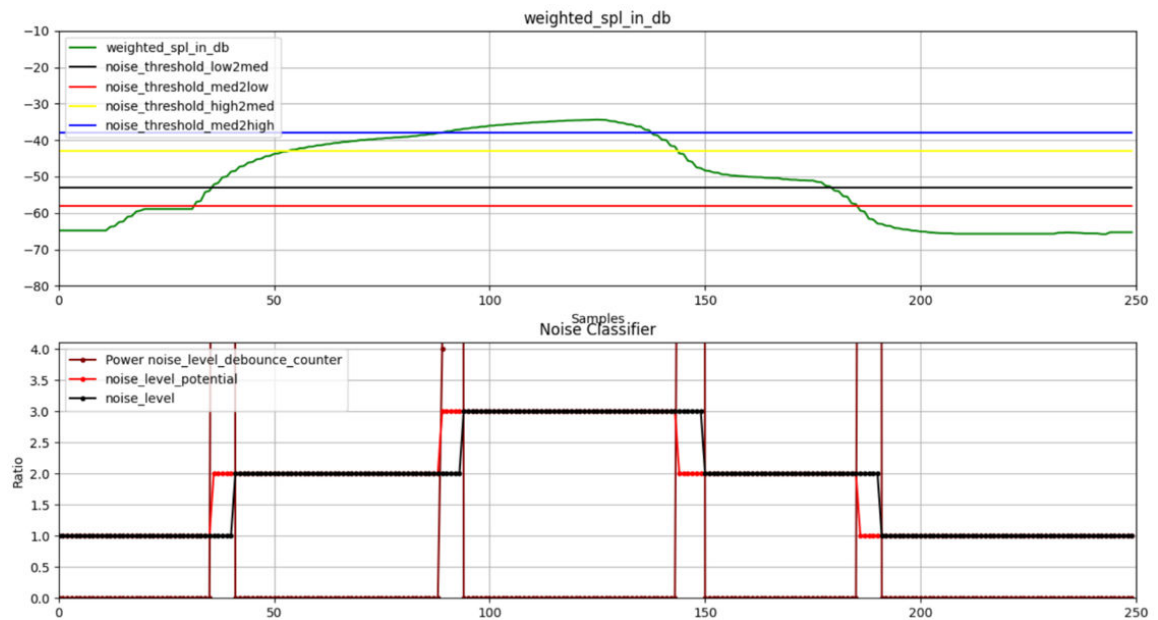


Figure 11-32 NoisID capability for based classification from the logger tool

- The noise category does not adjust if non-stationary noise is detected. The detection of non-stationary noise is carried out using an Energy Detector (ED), like that used in the Adaptive ANC feature.

11.6.1 Build Earbud app with NoisID

To build Earbud application with NoisID feature:

- Open earbud project in MDE.
- Add the following definitions to the project:
 - ENABLE_ANC
 - ENABLE_ADAPTIVE_ANC
 - ENABLE_ENHANCED_ANC
 - ENABLE_ANC_NOISE_ID

NOTE Other capability flags may need to be added depending on the use case.

 - ENABLE_ANC_NOISE_ID_3_LEVEL_CLASSIFY. This enables the noise categorization based on noise intensity.
- Add the `download_noise_id.edkcs` file in the `ro_fs` folder to add the NoisID capability.
- Type-based classification requires at least two adaptive ANC (AANC) modes to be configured for the NoisID to work. Similarly, level-based classification requires three Adaptive ANC modes corresponding to low, medium and high intensities.

5. Atleast two adaptive ANC (AANC) modes needs to be configured for the NoiseID to work:
 - a. Configure AANC modes in the `anc_config.c` file.
 - b. Set different noise ID categories for two modes.

An example to show the AANC configurations.

```
{ANC_CONFIG_MODE_ADAPTIVE, ANC_CONFIG_MODE_NOISE_CANCELLATION, TRUE,
ANC_NOISE_ID_CATEGORY_A},/*anc_mode_1*/ /*Adaptive ANC mode 1*/
{ANC_CONFIG_MODE_ADAPTIVE, ANC_CONFIG_MODE_NOISE_CANCELLATION, TRUE,
ANC_NOISE_ID_CATEGORY_B}, /*anc_mode_2*/ /*Adaptive ANC mode 2*/
{ANC_CONFIG_MODE_ADAPTIVE, ANC_CONFIG_MODE_NOISE_CANCELLATION, TRUE,
ANC_NOISE_ID_CATEGORY_C}, /*anc_mode_6*/ /*Adaptive ANC mode 2*/
```

6. Build and deploy.

11.6.2 Pydbg commands for NoiseID

This chapter lists the Pydbg commands for noiseID.

Table 11-18 Pydbg commands for NoiseID

Command	Description
<code>apps1.fw.call.EarbudTest_EnableNoiseId()</code>	Enables NoiseID.
<code>apps1.fw.call.EarbudTest_DisableNoiseId ()</code>	Disables NoiseID.
<code>apps1.fw.call.EarbudTest_IsNoiseIdEnabled ()</code>	Verifies if NoiseID is enabled.
<code>apps1.fw.call.EarbudTest_GetCurrentNoiseID ()</code>	Gets the current noise category.
<code>apps1.fw.call.EarbudTest_SetCurrentNoiseID ()</code>	Sets the NoiseID to a certain noise category.
<code>apps1.fw.call.EarbudTest_GetCurrentNoiseLevel ()</code>	Gets the current noise intensity level.

11.6.3 Tune NoiseID

To tune NoiseID capability:

1. Connect to QACT.
2. Launch AANC logger tool for graphical view of frequency bins, threshold, and ED event log.

NOTE For more information, see AANC logger tool section.

3. Insert the DUT in HATS.
4. Play each of the two types of noise separately from external speakers and obtain the power ratio of each noise category using one of the following methods:
 - AANC logger tool
 - QACT Live statistics

5. Set the power ratio threshold parameters based on the values.
6. Tune the tuning parameters.
7. Save the HTF and add the tuning parameters to `aanc_parameters.htf` in Earbud application

Table 11-19 Tuning parameters for type-based classification

Parameter name	Description	Tuning suggestion
Power ratio threshold 0	Power ratio threshold for switching to Noise category 0.	This should have sufficient distance from power ratio threshold 1 to not cause frequent and unwanted switching to the other noise category.
Power ratio threshold 1	Power ratio threshold for switching to Noise category 1.	This should have sufficient distance from power ratio threshold 0 to not cause frequent and unwanted switching to the other noise category.
Filter attack time	Attack time duration for smoothing filter of bin power.	Defines the speed of tracking non-stationary environmental changes. The filter attack time removes these disturbances from the estimate, avoiding noise category switching. This takes longer to recognize the new category of noise.
Filter decay time	Decay time duration for smoothing filter of bin power.	The filter decay time allows tracking changes when the environmental noise goes from a large amplitude to a lower one.
Noise hold timer	The time for which the current power ratio should meet or exceed the power ratio threshold in order to switch to the other noise category.	The value should be selected in such a way that frequent and unwanted noise category switch is avoided.
Energy detector parameters	The parameters and tuning of ED is same as the ED in AANC. For more information, see AANC section.	

Table 11-20 Tuning parameters for level-based classification

Parameter name	Description	Tuning suggestion
Noise threshold low2med	Noise threshold while switching from low noise level to medium noise level.	This value should be higher than Noise threshold med2low and remain lower than Noise threshold high2med and Noise threshold med2high.
Noise threshold med2low	Noise threshold while switching from medium noise level to low noise level.	This value should be lower than all thresholds.
Noise threshold high2med	Noise threshold while switching from high noise level to medium noise level.	This value should be higher than Noise threshold low2med and Noise threshold med2low and remain lower than Noise threshold med2high.
Noise threshold med2high	Noise threshold while switching from medium noise level to high noise level	This value should be higher than all thresholds.

Table 11-20 Tuning parameters for level-based classification (cont.)

Parameter name	Description	Tuning suggestion
Noise attack time	Attack time duration for smoothing filter of power.	The sum of weighted powers for each frame is calculated and then smoothed across frames using attack time to prevent frequent changes in intensity. The attack time monitors changes when the computed environmental power is on an increasing trend.
Noise decay time	Decay time duration for smoothing filter of power.	The decay time tracks changes when the computed environment power is in decreasing phase.
Debounce window size	The time for which the current power should meet or exceed the given power thresholds in order to switch to the other noise level category.	The value should be selected in such a way that frequent and unwanted noise category switch is avoided.
Weighting index	Weighting index to select one of the predefined weighting factors	The predefined weighting factors include <code>default_weighting</code> , <code>a_weighting</code> , <code>fb_weighting</code> , and <code>hy_weighting</code> . The weighting index assists in selecting one of them.
Energy detector parameters	The parameters and tuning of ED is same as the ED in AANC. For more information, see AANC section.	-

NOTE In level-based classification, four thresholds are used to differentiate noise into three categories. This prevents frequent transitions between noise levels. Higher thresholds are used when the noise intensity is increasing and lower thresholds are used when the intensity is decreasing.

A ANC stream APIs

The ANC stream APIs, such as Config ANC gain, Config ANC filter coefficients, Enable/ disable ANC, and Config Endpoint for ANC is used in audio capabilities to configure the ANC hardware and found at <ADK path>adk\src\libs\anc\ ANC lib files location.

The ANC stream APIs must be in `stream_for_anc.h` which can be found at `kymera\components\stream`. In ANC lib, `anc_configure.c` and `anc_configure_coefficients.c` files contain most of the functions required for ANC hardware configurations.

For example:

Stream key/API	Description
setIirCoefficients	Config IIR filter coefficients for each ANC channel by calling this trap API: <code>AudioAncFilterIirSet</code>
setLpfCoefficients	Config LPF filter coefficients for each ANC channel by calling this trap API: <code>AudioAncFilterLpfSet</code>
setDcFilters	Config DC filters using the trap API: <code>SourceConfigure</code>
setSmallLpf	Config small LPF filter using the trap API: <code>SourceConfigure</code>

Configure ANC gain

Stream key/API	Description	Example
<code>stream_anc_set_anc_fine_gain</code> (<code>STREAM_ANC_INSTANCE</code> anc_instance, <code>STREAM_ANC_PATH</code> path_id, <code>uint16</code> gain);	Configure the ANC path fine gains (background gains). anc_instance: ANC instance ID. For example, ANC0, ANC1. path_id: ANC input path ID. For example, FFA, FFB, and FB. gain: Gain applied to filter (8 bit linear gain value).	Set fine gain (background gain) for ANC instance 0 and path FFA: <code>stream_anc_set_anc_fine_gain</code> (<code>STREAM_ANC_INSTANCE_ANC0_ID</code> , <code>STREAM_ANC_PATH_FFA_ID</code> , <code>gain_value</code>)
<code>stream_anc_set_anc_foreground_fine_gain</code> (<code>STREAM_ANC_INSTANCE</code> anc_instance, <code>STREAM_ANC_PATH</code> path_id, <code>uint16</code> gain);	Configure the ANC path fine gains (foreground gains). anc_instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, and FB. gain: Gain applied to filter (8 bit linear gain value).	Set fine gain (foreground gain) for ANC instance 0 and path FFA: <code>stream_anc_set_anc_foreground_fine_gain</code> (<code>STREAM_ANC_INSTANCE_ANC0_ID</code> , <code>STREAM_ANC_PATH_FFA_ID</code> , <code>gain_value</code>)

Stream key/API	Description	Example
<pre>stream_anc_update_background_gains (STREAM_ANC_INSTANCE_MASK anc_instance_mask);</pre>	Copy the foreground gains to the background gains. anc_instance: ANC instance ID, for example, ANC0, ANC1, and ANC01. NOTE Gains for the FFA, FFB, and FB LPF are shadowed. The gain shifts are not shadowed.	Copy the foreground gains to the background gains for ANC instance 0 in ANC hardware manager capability: <pre>stream_anc_update_background_gains (AHM_ANC_INSTANCE_ANC0_ID)</pre>
<pre>stream_anc_set_anc_coarse_gain (STREAM_ANC_INSTANCE anc_instance, STREAM_ANC_PATH path_id, uint16 shift);</pre>	Configure the ANC path coarse gain (gain exponent). anc_instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, FB. gain: Gain applied to filter (4 bit shift value).	Set coarse gain for ANC instance 0 and path FFA: <pre>stream_anc_set_anc_coarse_gain (STREAM_ANC_INSTANCE_ANC0_ID, STREAM_ANC_PATH_FFA_ID, gain_shift_value)</pre>
<pre>stream_anc_get_anc_fine_gain (STREAM_ANC_INSTANCE anc_instance, STREAM_ANC_PATH path_id, uint16 *gain);</pre>	Get ANC path fine gain anc_instance: ANC instance ID, for example, ANC0, ANC1. path_id ANC: input path ID, for example, FFA, FFB, FB. gain: gain pointer to gain.	Get the current fine gain in ANC instance 0 and path FFA: <pre>uint16 instance0_current_gain; stream_anc_get_anc_coarse_gain (STREAM_ANC_INSTANCE_ANC0_ID, STREAM_ANC_PATH_FFA_ID, &instance0_current_gain);</pre>
<pre>stream_anc_get_anc_coarse_gain (STREAM_ANC_INSTANCE anc_instance, STREAM_ANC_PATH path_id, uint16 *shift);</pre>	Get ANC path coarse gain (gain exponent). anc_instance: ANC instance ID, for example, ANC0, ANC1). path_id ANC: input path ID, for example, FFA, FFB, FB. shift pointer: to gain shift applied to filter (4 bit shift value).	Get the current coarse gain in ANC instance 0 and path FFA: <pre>uint16 instance0_current_gain; stream_anc_get_anc_coarse_gain (STREAM_ANC_INSTANCE_ANC0_ID, STREAM_ANC_PATH_FFA_ID, &instance0_current_gain);</pre>

Configure ANC filter coefficients

Stream key/API	Description
<code>stream_anc_get_filters_coeff_number</code> (<code>STREAM_ANC_PATH</code> path);	Return the number of coefficient of a filter in a specified path of an ANC block Path: path to query path_id, for example, FFA, FFB, FB.
<code>stream_anc_set_anc_iir_coeffs</code> (<code>STREAM_ANC_INSTANCE</code> instance, <code>STREAM_ANC_PATH</code> path_id, unsigned num_coeffs, const <code>STREAM_ANC_IIR_COEFF_TYPE</code> *coeffs);	Configure an ANC IIR filter (sets foreground and background coefficients) Instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, FB. num_coeffs: Number of coefficients. Coeffs: Pointer to an array of IIR coefficients.
<code>stream_anc_set_anc_iir_foreground_coeffs</code> (<code>STREAM_ANC_INSTANCE</code> instance, <code>STREAM_ANC_PATH</code> path_id, unsigned num_coeffs, const <code>STREAM_ANC_IIR_COEFF_TYPE</code> *coeffs);	Configure an ANC IIR filter (sets the foreground coefficients) instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, FB. num_coeffs: Number of coefficients. coeffs: Pointer to an array of IIR coefficients.
<code>stream_anc_get_anc_iir_coeffs</code> (<code>STREAM_ANC_INSTANCE</code> instance, <code>STREAM_ANC_PATH</code> path_id, unsigned num_coeffs, <code>STREAM_ANC_IIR_COEFF_TYPE</code> *coeffs);	Get ANC IIR filter (gets the coefficients). Instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, FB. num_coeffs: Number of coefficients. Coeffs: Pointer to an array of IIR coefficients.
<code>stream_anc_update_background_iir_coeffs</code> (<code>STREAM_ANC_INSTANCE_MASK</code> anc_instance_mask);	update coefficients from foreground to background anc_instance_mask: ANC instance ID, for example, ANC0, ANC1.
<code>stream_anc_select_active_iir_coeffs</code> (<code>STREAM_ANC_INSTANCE_MASK</code> anc_instance_mask, <code>STREAM_ANC_BANK</code> coeff_set);	Configure ANC active IIR coefficients. anc_instance: ANC instance ID, for example, ANC0, ANC1. coeff_set: Whether banks are being used as foreground or background.
<code>stream_anc_set_anc_lpf_coeffs</code> (<code>STREAM_ANC_INSTANCE</code> instance, <code>STREAM_ANC_PATH</code> path_id, uint16 shift1, uint16 shift2);	Configure an ANC LPF filter (sets the LPF coefficients) instance: ANC instance ID, for example, ANC0, ANC1. path_id: ANC input path ID, for example, FFA, FFB, FB. shift1: Coefficient 1 expressed as a shift. shift2: Coefficient 2 expressed as a shift.

Enable/ disable ANC

Stream key/API	Description
<pre>stream_anc_enable (STREAM_ANC_INSTANCE stream_anc_instance_number, uint16 *anc_enable, bool (*resp_callback) (void));</pre>	Wrapper to enable or disable ANC with license check from ANC tuning capability stream_anc_instance_number: The number of ANC instances or blocks can be used. anc_enable: Bitfield controlling instance ANC0 and ANC1 signal paths. ANC enable Array of 1 used to control instance ANC0 in Earbud software since mono channel. Array of 2 used to control instance ANC0 and ANC1 in Headphone software since stereo channel. resp_callback: Callback function pointer for sending the response.
<pre>stream_get_anc_enable (uint16 *anc_enable_0_ptr, uint16 *anc_enable_1_ptr);</pre>	Retrieve the ANC enable or disable state anc_enable_0_ptr: Pointer to result (bitfield controlling instance ANC0 signal paths). anc_enable_1_ptr: Pointer to result (bitfield controlling instance ANC1 signal paths).

Configure Endpoint for ANC

Stream key/API	Description
<pre>stream_anc_configure_input (ENDPOINT *ep, STREAM_ANC_PATH path);</pre>	Associate a source endpoint with an ANC path. ep: Pointer to endpoint to configure. path: ANC input path ID, for example, FFA, FFB, FB.
<pre>stream_anc_configure_instance (ENDPOINT *ep, STREAM_ANC_INSTANCE instance);</pre>	Associate a source endpoint with an ANC instance. ep: Pointer to endpoint to configure. Instance: ANC instance ID, for example, ANC0, ANC1.
<pre>stream_anc_configure_gain (ENDPOINT *ep, STREAM_ANC_PATH path, unsigned value);</pre>	Associate a source endpoint with an ANC path and set the fine gain. ep: Pointer to endpoint to configure. path: ANC input path ID, for example, FFA, FFB, FB. Value: gain value.
<pre>stream_anc_configure_gain_shift (ENDPOINT *ep, STREAM_ANC_PATH path, unsigned value);</pre>	Associate a source endpoint with an ANC path and set the coarse gain ep: Pointer to endpoint to configure. path: ANC input path ID, for example, FFA, FFB, FB). Value:
<pre>stream_anc_configure_dc_filter_enable (ENDPOINT *ep, STREAM_ANC_PATH path, unsigned value);</pre>	Enable DC filter for an ANC path ep: Pointer to endpoint to configure. path: ANC input path ID, for example, FFA, FFB, FB. Value: dc filter enable value.

Stream key/API	Description
<code>stream_anc_configure_dc_filter_shift</code> (ENDPOINT *ep, STREAM_ANC_PATH path, unsigned value);	Set the DC filter shift for an ANC path ep: Pointer to endpoint to configure. path: ANC input path ID, for example, FFA, FFB, FB. Value: the value of DC filter shift.
<code>stream_anc_enable_sdm</code> (ENDPOINT *endpoint);	Enables the Sigma-Delta Modulator on the feedback tuning output Endpoint: Pointer to the endpoint.
<code>stream_anc_connect_feedback_monitor</code> (ENDPOINT *endpoint, ANC_FBMON_SRC source);	Connect Feedback monitor to an IIR filter input. endpoint: Pointer to the endpoint. source: Which IIR filter input to connect. ANC_FBMON_SRC source options: ANC_FBMON_FFA = 0, Feed-forward Filter (typical outside microphone) ANC_FBMON_FB = 1, Feedback Filter ANC_FBMON_ANC_OUT = 2, ANC Output

Connect endpoints to FBMON of ANC

Stream key/API	Description	Example
<code>stream_anc_source_configure</code> (ENDPOINT *ep, STREAM_CONFIG_KEY key, uint32 value)	A CODEC or DIGIMIC type endpoint can be connected to FBMON, to monitor one of FFA or FB ANC out signals. ep: pointer to the endpoint for which FBMON to be connected. key: stream key 0x1112 to configure the decimation chain. value: mask of the decimation chain in the higher 16-bits and the decimation number to be selected in the lower 16-bits.	Codec instances 0,1 and 2 are supported for FBMON via the CODEC or DIGIMIC endpoint. This means only the decimation chains 0,1,2,3,4, and 5 are supported. For example, the FBMON of ANC instance 0 is to be connected to decimation chain 1, then the value parameter needs to be formatted as: value = (0x7800 << 16) value += (0x0001 << 11) 0x7800 is the mask to select the decimation chain and it is moved 16 bits left to place it in the higher 16-bits. 0x0001 is the decimation chain number and it is shifted left by 11 because the mask starts at the 11th bit. This is placed in the lower 16-bits. <code>stream_anc_source_configure(ep, 0x1112, value)</code>

Terms and definitions

Term	Definition
ACCMDS	Audio Client Commands
ADC	Analog to Digital Converter
ADK	Audio Development Kit
ANC	Active Noise Cancellation
API	Application Programming Interface
ATR	Auto Transparency
CAPID	Capability ID
cVc	Clear Voice Capture
DAC	Digital to Analog Converter
DC	Direct Current
DFU	Device Firmware Upgrade
DUT	Device Under Test
FB	Feedback
FF	Feedforward
GUI	Graphical User Interface
I ² S	Inter-Integrated Circuit Sound
IIR	Infinite Impulse Response (digital filter design and signal processing)
LED	Light Emitting Diode
LIBS	Libraries
MDE	Multi-Core Development Environment
MFB	Multi Function Button
Mic	Microphone
MP	Micro Processor
PIO	Programmable Input Output
QACT	Qualcomm Audio Calibration Tool
QTI	Qualcomm Technologies International, Ltd.
Rx	Receive or Receiver
SPI	Serial Peripheral Interface
UCID	Use Case Identifier

Term	Definition
USB	Universal Serial Bus
VAD	Voice Activity Detection
WND	Wind Noise Detect

Document references

Document	Reference
<i>Kymera Capability Library User Guide</i>	80-CF976-1
<i>QCC512x and QCC302x/3x Development Kit and ADK 6.4.x Quick Start User Guide</i>	80-CH108-1
<i>QCC512x_QCC302x.SRC.1.0 Earbud Quick Start User Guide</i>	80-CH145-1
<i>QCC514x_QCC304x.SRC.1.0 Earbud Quick Start User Guide</i>	80-CH213-1
<i>Qualcomm Kymera Tools User Guide</i>	80-15677-1
<i>End to End Wireless Debug User Guide</i>	80-16570-1
<i>Device Test Service User Guide</i>	80-14839-1

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this “Material”), is subject to your (including the corporation or other legal entity you represent, collectively “You” or “Your”) acceptance of the terms and conditions (“Terms of Use”) set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. (“Qualcomm Technologies”), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as “Qualcomm Internal Use Only”, no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies’ prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.